ПРОГРАММИРОВАНИЕ на АППАРАТНОМ УРОВНЕ

2-е издание

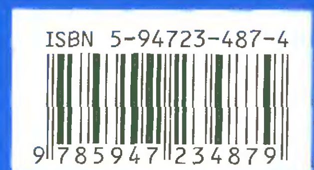
Специальный справочник

В этой книге рассматриваются полезные, но не описанные в общедоступной литературе возможности персональных IBM-совместимых компьютеров. Точнее, эти возможности обычно описаны наполовину: потребителю предоставляется документация, содержащая сведения о конструкции устройства, но не дается никаких рекомендаций по его применению на практике... Технология изучения и применения особенностей аппаратуры, которая предлагается вашему вниманию, основана на совместном использовании возможностей PCI BIOS, VESA BIOS, а также метода линейной адресации данных. Информация, собранная в книге, представляет интерес прежде всего для тех, кто работает на границах спектра программного обеспечения и не удовлетворен возможностями, предоставляемыми стандартными драйверами и операционными системами.
Во второе издание книги внесены исправления и дополнения. В частности, добавлены две главы, в одной из которых рассматривается работа с хост-контроллером и устройствами USB на уровне аппаратуры, а в другой — особенности программирования NE2000-совместимых адаптеров Ethernet.



на прилагаемой дискете находятся тексты программ из книги

Владимир Кулаков


ISBN 5-94723-487-4
9 785947 234879

Информация, которую вы найдете в справочнике:

- клавиатурные функции BIOS
- недокументированные возможности процессоров
- аппаратно-независимое управление устройствами PCI
- функции управления режимами работы видеоконтроллеров
- управление дисковыми накопителями на уровне BIOS
- работа с хост-контроллером и устройствами USB на уровне аппаратуры
- программирование NE2000-совместимых адаптеров Ethernet

и многое другое...

Уровень пользователя

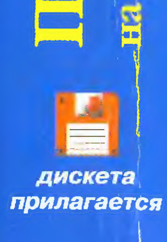
Начинающий Опытный Профессионал



2-е издание

ПРОГРАММИРОВАНИЕ на АППАРАТНОМ УРОВНЕ

Специальный справочник



дискета прилагается

Владимир Кулаков

Нужная информация всегда под рукой!

Владимир Кулаков

- *аппаратно-независимое управление устройствами*
- *программная обработка прерываний*
- *работа с хост-контроллером и устройствами USB*
- *программирование NE2000-совместимых адаптеров Ethernet*





текcты программ из книги на дискете

ПРОГРАММИРОВАНИЕ на АППАРАТНОМ УРОВНЕ

2-е издание

Специальный справочник



СЕРИЯ

Специальный справочник



Владимир Кулаков

ПРОГРАММИРОВАНИЕ на АППАРАТНОМ УРОВНЕ

Специальный справочник



Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара
Киев · Харьков · Минск
2003

Владимир Кулаков

Программирование на аппаратном уровне: специальный справочник. второе издание (+дискета)

Главный редактор	*Е. Строганова*
Заведующий редакцией	*И. Корнеев*
Руководитель проекта	*А. Васильев*
Литературный редактор	*Е. Ваулина*
Художник	*Н. Биржаков*
Корректор	*А. Моносов*
Верстка	*А. Дорошенко*

ББК 32.973.23я22
УДК 681.3(083)

Кулаков В.

К90 Программирование на аппаратном уровне: специальный справочник (+дискета). 2-е издание. — СПб.: Питер, 2003. — 847 с.: ил.

ISBN 5-94723-487-4

В книге рассматриваются возможности персональных IBM-совместимых компьютеров, рекомендации по использованию которых не даются в официальной технической документации. Информация, собранная в книге, интересна прежде всего для тех, кто не удовлетворен возможностями, предоставляемыми стандартными драйверами и операционными системами. С одной стороны, это создатели «несерьезных», но сложных программ — компьютерных игр, а с другой — разработчики самого серьезного обеспечения, предназначенного для систем управления разнообразными техническими объектами.

Во второе издание книги внесены исправления и дополнения. В частности, добавлены две главы, в одной из которых рассматривается работа с хост-контроллером и устройствами USB на уровне аппаратуры, а в другой — особенности программирования NE2000-совместимых адаптеров Ethernet.

Неотъемлемой частью издания является дискета с текстами программ, листинги которых представлены в книге.

ISBN 5-94723-487-4

ООО «Питер Принт», 196105, Санкт-Петербург, ул. Благодатная, д. 67в.
Лицензия ИД № 05784 от 07.09.01.
Налоговая льгота – общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 – литература учебная.
Подписано в печать 30.06.03. Формат 60×90/16. Усл. п. л. 68,37.
Доп. тираж 3000 экз. Заказ № 231.
Отпечатано с готоаых диапозитивов в ООО «Типография Правда 1906».
191119, С.-Петербург, Социалистическая ул., 11-а.

Краткое содержание

Содержание

Введение

В книге, которую я предлагаю вашему вниманию, рассматриваются некоторые полезные, но не описанные в общедоступной литературе возможности персональных IBM-совместимых компьютеров. Точнее, эти возможности обычно описаны наполовину: потребителю предоставляется документация, содержащая сведения о конструкции устройства (datasheet), но не дается никаких рекомендаций по его применению на практике (отсутствуют описания application notes). Я хотел бы по мере сил восполнить этот пробел.

Аппаратные средства в современных персональных компьютерах настолько сильно связаны между собой, что начинающий программист обычно не знает, с чего начать. Прежде чем приступить к работе, он вынужден изучить и запомнить огромный объем разнообразных сведений, что сильно осложняет обучение. Я рекомендую в процессе работы использовать приемы, описанные в Programmer's Journal (хороший был журнал...), — они позволяют осваивать особенности различных компонентов компьютера поэтапно, по частям.

Технология изучения и применения особенностей аппаратуры, которую я предлагаю вашему вниманию, основана на совместном использовании возможностей PCI BIOS, VESA BIOS, а также метода линейной адресации данных. Информация, собранная в книге, представляет интерес прежде всего для тех, кто работает на границах спектра программного обеспечения и не удовлетворен возможностями, предоставляемыми стандартными драйверами и операционными системами. С одной стороны, это создатели несерьезных, но сложных программ — компьютерных игр, а с другой — разработчики самого серьезного обеспечения, предназначенного для систем управления разнообразными техническими объектами.

Недокументированные возможности аппаратуры

К сожалению, многие возможности современной (разработанной в течение последних десяти лет) аппаратуры персональных компьютеров являются недокументированными (undocumented). Сам термин «недокументированные возможности» не следует воспринимать слишком буквально — на самом деле документация всегда где-то существует, разработчики аппаратуры, во всяком случае, ее имеют. Проблема заключается в том, что по многим причинам они крайне неохотно делятся информацией с потребителями. Первая из этих причин — ничем не мотивированная, совершенно иррациональная жадность. Скрывать информацию о своих изделиях не только не выгодно, но часто и весьма опасно для фирмы: с точки зрения потребителей, закрытость — это антиреклама, а с точки зрения конкурентов — уязвимое место. История вычислительной техники хранит примеры того, как гиганты компьютерной промышленности (Apple, DEC, IBM, Motorola) теряли целые сегменты мирового рынка по причине неполного или несвоевременного предоставления информации потребителям.

Например, фирма IBM подбирала микропроцессор для своих первых персональных компьютеров именно по признаку доступности документации. В свою очередь, компьютеры типа IBM PC/AT также получили широкое распространение благодаря тому, что написаны и изданы тысячи книг об их внутренней организации и программном обеспечении, в которых подробно разбирается буквально каждая деталь. А теперь оцените убытки, которые понесли (и продолжают нести по сей день) конкуренты фирм IBM и Intel из-за того, что их документация когда-то оказалась менее доступной! Внедрение свободного доступа к технической документации через Интернет показывает, что руководители компьютерных фирм учли печальный опыт.

Можно привести и противоположный пример, связанный с IBM: руководство фирмы допустило ошибку, когда, опасаясь конкуренции, решило «зажать» новую шинную архитектуру MCA и не предоставлять другим фирмам лицензий на ее использование. В результате вместо устранения конкурентов IBM утратила контроль над рынком персональных компьютеров и потеряла позиции лидера. Фирма Intel, внедряя шинную архитектуру PCI, сделала выводы из опы-

та IBM и фактически навязала свою архитектуру всем (в том числе — конкурентам), полностью вытеснив стандарт VLB.

Вторая причина ограничения доступа к информации — страх разработчиков перед критикой, вызванной допущенными в конструкции выпускаемых изделий недочетами и ошибками. Линия персональных компьютеров, ведущая свое происхождение от машин IBM PC, наследует все непродуманные решения и дефекты, допущенные инженерами-проектировщиками за двадцать лет ее существования. Радиолюбительские трюки, которые позволяют разработчикам оборудования сэкономить несколько дискретных элементов, могут полностью блокировать или необычайно затруднить дальнейшее развитие системы. Платить за решение всех проблем, однако, в конечном итоге приходится потребителям.

Третья причина — боязнь, что конкуренты используют полученные сведения для создания аналогичных изделий (клонов). Но в современном мире очень трудно защитить свои секреты от сильных противников — крупных корпораций или иностранных государств (советская разведка, например, весьма активно похищала технологии у DEC и IBM). В результате от сокрытия информации страдают только легальные потребители продукции.

Четвертая причина — стремление сотрудников фирмы сохранить монополию на производство программного обеспечения для изготавливаемой аппаратуры. Отдельным лицам такая монополия выгодна, а фирме в целом часто наносит серьезный экономический и моральный ущерб. Компьютерная промышленность во многом развивается за счет энтузиастов: прототипы современных персональных компьютеров и наиболее известных программных продуктов для них созданы отдельными лицами или небольшими группами из двух-трех человек. При жестком ограничении доступа к информации в первую очередь страдают как раз подобные энтузиасты, а фирма сама себе перекрывает приток свежих кадров, идей и технологий.

Специальные возможности аппаратных средств

Общедоступная документация для выполнения каждой операции с аппаратурой предлагает, как правило, единственный способ, признанный наилучшим для массового применения в заданной области.

Однако прием, идеальный в одном случае, будет заведомо неоптимальным в любом другом — это издержки оптимизации. Если ситуация отличается от типовой, то приходится искать нестандартные решения, то есть заниматься научно-исследовательской деятельностью и восполнять недостаток знаний путем экспериментов.

Существующие операционные системы рассчитаны на массовое применение в офисе и в бытовых условиях. При этом нельзя предсказать, какое программное обеспечение, в какой комбинации и при какой конфигурации аппаратуры будет использовать потребитель. В результате основные силы и средства операционная система массового применения расходует не на полезную работу, а на согласование между собой разномастной аппаратуры и разнообразных программ.

Для систем промышленного и военного применения, систем управления транспортом, медицинских систем, электронных тренажеров ситуация может быть совершенно иной: в этом случае на оборудова-нии, имеющем строго определенную конфигурацию, решается строго определенный набор задач [15, 24]. Соответственно можно заранее распределить все ресурсы, а у операционной системы остаются две основные функции — защита от сбоев и загрузка программ. Быстродействие специализированной системы намного выше, чем универсальной (за счет специализации), но программист в этом случае вынужден работать непосредственно с аппаратурой.

Даже в обычных бытовых компьютерах прямая работа с аппаратурой очень часто является единственно возможным способом обеспечения необходимого быстродействия. Например, операция вывода точки на экран монитора заключается в том, что процессор заносит байт кода цвета в память видеоконтроллера (для этого нужно выполнить одну команду пересылки данных MOV). Выполнение аналогичной операции через прерывания MS-DOS происходит медленнее в 100–1000 раз (процедура обработки прерывания включает сотни команд), что совершенно недопустимо при выводе динамических изображений (например, в играх).

Принятая у изготовителей компонентов для бытовых компьютеров система «делай что хочешь» приводит к тому, что со многими устройствами работать непосредственно (без драйверов) невозможно — нет единого отраслевого стандарта на интерфейс. Однако на компоненты двойного назначения (микропроцессоры, дисковые накопители, видеоконтроллеры), которые используются как в бытовых, так и в промышленных системах, стандарты достаточно жесткие, что упрощает разработку и отладку программ.

Иногда аппаратура стандартизирована лучше, чем программное обеспечение, предназначенное для работы с ней, как это произошло с манипуляторами типа мышь: фактически имеется единый стандарт на протокол передачи данных через последовательный порт, а вот драйверы каждый из изготовителей реализовывал (в рекламных целях) с определенными нестандартными особенностями. В результате MS Windows, например, обычно работает с мышью непосредственно, а не через драйверы изготовителей.

Необходимость в работе с аппаратурой возникает также при появлении нового оборудования, для которого еще нет готовых драйверов, или новой операционной системы, для которой драйверы нужно полностью переписывать. Так, распространение системы Linux вынудило многие фирмы открыть доступ к ранее закрытым разделам информации, а программистов — заняться изучением работы разнообразных периферийных устройств.

Следует учитывать, что у современной аппаратуры имеется множество недокументированных возможностей, однако далеко не все из них целесообразно применять. Некоторые свойства и особенности аппаратуры не были документированы именно потому, что совершенно бесполезны или опасны. Например, некорректная установка параметров видеоконтроллера может вывести из строя или сам контроллер, или подключенный к нему монитор (даже современные мониторы не полностью защищены от угрозы такого типа). Неправильное использование команды записи данных или форматирования жесткого диска может привести не только к потере всей имеющейся на нем информации, но и к физическому повреждению диска.

Полезные возможности отличаются следующими свойствами:

- стандартизированностью (не являясь де-юре стандартом, эти возможности де-факто реализованы всеми изготовителями данного типа оборудования);
- эффективностью (дают явный выигрыш в быстродействии);
- безопасностью;
- простотой.

Фактор стандартизированности играет важную роль при разработке систем широкого применения, которые могут комплектоваться оборудованием разных изготовителей. Фактор эффективности важен для систем управления, тренажеров и компьютерных игр. Факторы безопасности и простоты существенны для самих программистов.

Порядок изложения материала в книге

Предлагаемая вашему вниманию книга ориентирована на подготовленного пользователя, который уже ознакомился по крайней мере с основами работы на персональном компьютере и правилами программирования на языке ассемблера процессоров серии x86. Знание ассемблера совершенно необходимо при непосредственной работе с аппаратурой: использование языков высокого уровня только «замутняет» простые по своей сути операции. Даже аппаратно-ориентированный язык С может создавать определенные проблемы, связанные со средствами оптимизации, встроенными в компиляторы [1].

Порядок представления материала ориентирован на максимально быстрое включение читателя в процесс практического использования полученных знаний. Вначале рассматриваются средства, совершенно необходимые в процессе отладки программы: ввод данных с клавиатуры, линейная адресация в реальном режиме DOS, особенности работы с устройствами PCI, вывод информации на экран. Далее эти средства используются для наглядной демонстрации приемов работы с различными периферийными устройствами (мышью, дисководами и т. д.). К сожалению, объем документации по аппаратуре настолько велик, что при подборе материала я был вынужден ограничиться только теми устройствами, без которых реальная работа вообще невозможна.

Использование аппаратных средств в «чистом» виде — дело весьма трудоемкое, и обычно при программировании применяется некоторая комбинация из аппаратных и программных средств. Кроме того, работать непосредственно с аппаратурой имеет смысл только в том случае, если невозможно решить задачу средствами операционной системы — многие типы периферийных устройств недостаточно стандартизированы. Поэтому в начале каждой главы приводится справочный материал по средствам высокого уровня (функциям MS-DOS и BIOS), предназначенным для работы с рассматриваемыми в этой главе устройствами. Далее следуют рекомендации и пояснения, касающиеся внутренней организации и особенностей работы устройства, а также конкретные примеры выполнения различных операций.

Листинги всех примеров имеют простую линейную структуру, наглядно демонстрирующую порядок выполнения операций. Приме-

ры оформлены по единому шаблону, рассчитанному на использование упрощенного режима Ideal, введенного в Turbo Assembler фирмой Borland. Минимальные требования программ *к составу оборудования* указываются отдельно в каждом конкретном случае — может потребоваться какое-либо определенное устройство (например, мышь) или характеристики устройства должны быть не хуже заданных (например, класс процессора не ниже Pentium).

К книге прилагается *гибкий диск*, содержащий исходные коды и исполняемые модули программ. Исходные коды программ на ассемблере находятся в папке SOURCE, исполняемые модули — в папке EXECUTE, прототипы системных файлов для загрузочного диска — в папке SYSTEM (для MS-DOS 6.22 — в папке DOS_6_22, для DOS 7.0 — в папке WIN_9X).

ВНИМАНИЕ

Не рекомендуется запускать примеры с прилагаемого гибкого диска. Перед тем как начать работу с исходными кодами или исполняемыми модулями, скопируйте их на жесткий диск или специально созданный загрузочный диск.

Общие требования к аппаратуре и операционной системе

Видеоконтроллер должен поддерживать стандарт VESA по крайней мере версии 2.0.

Обязательное требование к *операционной системе*: должен быть запущен драйвер-русификатор, поскольку в примерах для диалога с оператором используются сообщения на русском языке.

Приведенные в книге программы рассчитаны на свободный доступ к памяти и аппаратуре, то есть должны запускаться в реальном (однозадачном) режиме работы MS-DOS или в режиме эмуляции MS-DOS, который имеется в Windows 95/98. Примеры, использующие линейную адресацию памяти, *конфликтуют* с драйверами EMM и QEMM, переводящими процессор в режим виртуальных машин; для запуска таких примеров на компьютере, где используется драйвер EMM-типа, нужно создавать специальный загрузочный диск.

Загрузочный диск создается как системный диск DOS. Можно либо сразу отформатировать диск как системный (используя команду

format с ключом /s), либо сделать его системным после форматирования при помощи команды sys. В первом случае команда форматирования выглядит так:

```
format a:/s
```

Во втором случае последовательно выполняются две операции:

```
format a:
sys a:
```

После этого требуется скопировать на гибкий диск файлы операционной системы, необходимые для запуска драйвера-русификатора, а также создать на нем файлы autoexec.bat и config.sys.

Файл config.sys должен содержать следующие строки:

```
COUNTRY = 07.866.COUNTRY.SYS
DEVICE = DISPLAY.SYS CON = (EGA,,1)
```

Содержимое файла autoexec.bat зависит от используемой версии операционной системы. Для MS-DOS 6.22 файл имеет вид:

```
MODE CON CP PREP = ((866) EGA.CPI)
MODE CON CP SEL = 866
KEYB RU,,KEYBRD2.SYS
```

Если диск создавался в MS-DOS 7.0 (Windows 9x), файл autoexec.bat будет содержать следующие строки:

```
MODE CON CP PREP = ((866) EGA3.CPI)
MODE CON CP SEL = B66
KEYB RU,,KEYBRD3.SYS
```

В случае, если загрузочный диск создавался под MSDOS 6.22, на него нужно переписать из папки C:\DOS следующие файлы:

- keyb.com;
- mode.com;
- ega.cpi;
- country.sys;
- dysplay.sys;
- driver.sys;
- keybrd2.sys.

ПРИМЕЧАНИЕ

Указанные файлы нужно записывать прямо в корневой каталог загрузочного диска.

В случае, если загрузочный диск создавался под Windows 9x, на него нужно переписать следующие файлы из папки C:\WINDOWS\COMMAND:

- keyb.com;
- mode.com;
- ega3.cpi;
- country.sys;
- dysplay.sys;
- driver.sys;
- keybrd3.sys.

Когда загрузочный диск создан, на него нужно переписать программы-примеры. После этого можно перезагрузить компьютер с гибкого диска и начать работу. Для запуска примера с загрузочного диска достаточно набрать имя примера (например, 1st_1_01) и нажать клавишу Enter.

ВНИМАНИЕ

Загрузочный диск не должен быть защищен от записи (в примерах из главы 6 файлы записываются прямо на него).

От издательства

Выражаем искреннюю признательность М. Гуку, Ю. Малашенкову и С. Казакову за ценные замечания и предложения, сделанные в ходе обсуждения материала этой книги.

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты comp@piter-press.ru (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на web-сайте издательства http://www.piter-press.ru.

Глава 1
Работа с клавиатурой

Клавиатура является основным средством ввода текстовой информации в компьютер, поэтому при программировании на низком уровне (на языке ассемблера) программист в первую очередь вынужден осваивать работу с клавиатурой: он должен изучить способы кодирования текстовых символов и особенности использования функций операционной системы. Если у программиста возникает потребность в работе на уровне аппаратного обеспечения, он должен также разобраться с особенностями программирования контроллера клавиатуры и контроллера прерываний.

Представление символов и управляющих кодов в памяти компьютера

Система представления символов в персональных компьютерах базируется на Американском стандартном коде для обмена информацией (American Standard Code for Information Interchange), который был введен в 1963 году и ставил в соответствие каждому символу семиразрядный двоичный код, обеспечивающий представление 128 символов. ASCII-код включал две группы символов:

- управляющие символы, используемые в коммуникационных протоколах для передачи команд периферийным устройствам;
- символы пишущей машинки — цифры, буквы и специальные знаки.

Управляющие символы имеют коды с номерами от 0 до 1Ah. К управляющим относится также символ с кодом 7Fh. Каждый управляю-

щий символ выполняет строго определенную функцию. Функции и кодовые обозначения управляющих символов описаны в табл. 1.1. Все остальные символы относятся к алфавитно-цифровой группе (группе символов пишущей машинки).

Таблица 1.1. Управляющие символы ASCII-кода

Код символв	Мнемоническое обозначение	Назначение символа
00h	NUL	Пустой символ
01h	SOH	Начало заголовка (начало блока данных)
02h	STX	Начало текста
03h	ETX	Конец текста
04h	EOT	Конец передачи
05h	ENQ	Запрос подтверждения
06h	ACK	Подтверждение
07h	BEL	Звонок (звуковой сигнал)
08h	BS	Забой (возврат на одну позицию влево)
09h	HT	Горизонтальная табуляция
0Ah	LF	Перевод строки
0Bh	VT	Вертикальная табуляция
0Ch	FF	Перевод формата (переход к новой странице)
0Dh	CR	Возврат каретки
0Eh	SO	Переход на нижний регистр
0Fh	SI	Переход на верхний регистр
10h	DLE	Завершение сеанса связи
11h	DC1	Управление устройством № 1
12h	DC2	Управление устройством № 2
13h	DC3	Управление устройством № 3
14h	DC4	Упрааление устройством № 4
15h	NAK	Ошибка передачи
16h	SYN	Холостой ход передатчика
17h	ETB	Конец передачи блока
18h	CAN	Отмена
19h	EM	Конец носителя данных
1Ah	SUB	Подстановка (замена символа)
1Bh	ESC	Переход (посылка сложной команды)
1Ch	FS	Разделитель файлов

продолжение ⇨

Таблица 1.1 (*продолжение*)

Код символа	Мнемоническое обозначение	Назначение символа
1Dh	GS	Разделитель групп
1Eh	RS	Разделитель записей
1Fh	US	Разделитель элементов
7Fh	DEL	Удаление символа

Чтобы отобразить символы европейских алфавитов и символы псевдографики, ASCII-код был расширен до 256 символов. Графическое представление символов расширенного ASCII-кода показано на рис. 1.1. Это так называемая американская кодировка (кодировка IBM), которая в операционных системах корпорации Microsoft носит также название «Кодовая страница 437».

Младшая цифра	Старшая цифра															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		►		0	@	P	`	p	Ç	É	á	░	└	╨	α	≡
1	☺	◄	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	ß	±
2	☻	↕	"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
3	♥	‼	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
5	♣	§	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6	♠	▬	&	6	F	V	f	v	å	û	ª	╢	╞	╓	µ	÷
7	•	↨	'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8	◘	↑	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	∙
A	◙	→	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
B	♂	←	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
C	♀	∟	,	<	L	\	l	|	î	£	¼	╝	╠	▄	∞	ⁿ
D	♪	↔	-	=	M	]	m	}	ì	¥	¡	╜	═	▌	φ	²
E	♫	▲	.	>	N	^	n	~	Ä	₧	«	╛	╬	▐	ε	■
F	☼	▼	/	?	O	_	o	⌂	Å	ƒ	»	┐	╧	▀	∩	

ASCII (0–7) Расширение ASCII (8–F)

Рис. 1.1. Представление символов ASCII-кода в американской кодировке

Однако по мере распространения персональных компьютеров постоянно возникала потребность в добавлении изображений новых символов, поэтому каждая страна мира сейчас имеет свою собственную кодовую страницу, а в многоязычных странах обычно применяется несколько таких страниц.

Младшая цифра	Старшая цифра															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		►		0	@	P	`	p	А	Р	а		└	╨	р	Ё
1	☺	◄	!	1	A	Q	a	q	Б	С	б	▒	┴	╤	с	ё
2	☻	↕	"	2	B	R	b	r	В	Т	в	▓	┬	╥	т	Є
3	♥	‼	#	3	C	S	c	s	Г	У	г	│	├	╙	у	є
4	♦	¶	$	4	D	T	d	t	Д	Ф	д	┤	─	╘	ф	Ї
5	♣	§	%	5	E	U	e	u	Е	Х	е	╡	┼	╒	х	ї
6	♠	▬	&	6	F	V	f	v	Ж	Ц	ж	╢	╞	╓	ц	Ў
7	•	↨	'	7	G	W	g	w	З	Ч	з	╖	╟	╫	ч	ў
8	◘	↑	(	8	H	X	h	x	И	Ш	и	╕	╚	╪	ш	°
9	○	↓	)	9	I	Y	i	y	Й	Щ	й	╣	╔	┘	щ	∙
A	◙	→	*	:	J	Z	j	z	К	Ъ	к	║	╩	┌	ъ	
B	♂	←	+	;	K	[	k	{	Л	Ы	л	╗	╦	█	ы	√
C	♀	∟	,	<	L	\	l	¦	М	Ь	м	╝	╠	▄	ь	№
D	♪	↔	-	=	M	]	m	}	Н	Э	н	╜	═	▌	э	¤
E	♫	▲	.	>	N	^	n	~	О	Ю	о	╛	╬	▐	ю	■
F	☼	▼	/	?	O	_	o	⌂	П	Я	п	┐	╧	▀	я	

Рис. 1.2. Представление символов ASCII-кода в русской кодовой странице 866

Представление символов ASCII-кода в русской кодовой таблице MS-DOS (кодовая страница 866) показано на рис. 1.2. Как видно из рисунка, символов в таблице гораздо больше, чем клавиш в алфавитно-цифровой части типовой клавиатуры, показанной на рис. 1.3 (101-клавишный AT-совместимый вариант исполнения), поэтому каждой клавише поставлено в соответствие несколько различных символов. ASCII-код, генерируемый при нажатии клавиши, определяется не только этой клавишей, но и состоянием управляющих клавиш Caps Lock и Shift, а также режимом работы драйвера клавиатуры, то есть текущим языком — русским или английским.



Рис. 1.3. Расположение клавиш на клавиатуре AT-типа

Кроме ASCII-кодов, для идентификации клавиш используются также скан-коды. Скан-коды в старых клавиатурах (появившихся до использования микроконтроллеров) являлись порядковыми номерами клавиш: нумерация велась сверху вниз, справа налево. С целью сохранения совместимости со старым программным обеспечением микропроцессоры современных клавиатур преобразуют действительные порядковые номера клавиш в номера, соответствующие латинской раскладке на клавиатуре IBM XT с учетом дополнительных клавиш клавиатуры IBM AT. В результате распределение номеров перестало быть строго упорядоченным. Кроме того, функции BIOS также выполняют перекодировку скан-кодов с целью упрощения анализа этих кодов в прикладных программах. Поэтому существует несколько различных видов таблиц скан-кодов:

- собственная внутренняя таблица встроенного микроконтроллера клавиатуры;
- таблица для обмена кодами между контроллером клавиатуры и специализированным клавиатурным микропроцессором системной платы;
- таблица кодов, которые клавиатурный микропроцессор передает подпрограммам BIOS;
- таблица кодов, которые BIOS передает прикладным программам.

При работе с функциями BIOS интерес представляет последняя из этих таблиц. Скан-коды BIOS для клавиш алфавитно-цифровой группы приведены в табл. 1.2; скан-коды клавиш функциональной, дополнительной и цифровой групп, а также комбинаций клавиш показаны в табл. 1.3. Следует учитывать, что клавиши цифровой группы, расположенной с правой стороны клавиатуры, могут использоваться не только как символьные (цифровые), но и как управляющие — в зависимости от состояния клавиши Num Lock.

Таблица 1.2. Скан-коды BIOS для клавиш алфавитно-цифровой группы

Скан-код	Режим		Скан-код	Режим	
Русский	Латинский	Русский	Латинский	Латинский	Русский
01h	Esc		1Eh	A и a	Ф и ф
02h	1 и !	1 и !	1Fh	S и s	Ы и ы
03h	2 и @	2 и «	20h	D и d	В и в
04h	3 и #	3 и №	21h	F и f	А и а
05h	4 и $	4 и ;	22h	G и g	П и п
06h	5 и %	5 и %	23h	H и h	Р и р

Скан-код	Режим		Скан-код	Режим	
Русский	Латинский	Русский	Латинский	Латинский	Русский
07h	6 и ^	6 и :	24h	J и j	О и о
08h	7 и &	7 и ?	25h	K и k	Л и л
09h	8 и *	8 и *	26h	L и l	Д и д
0Ah	9 и (	9 и (	27h	; и :	Ж и ж
0Bh	0 и)	0 и)	28h	" и «	Э и з
0Ch	– и _	– и _	29h	" и ~	Ё и ё
0Dh	= и +	= и +	2Ah	Левая клавиша Shift	
0Eh	Back Space		2Bh	\ и |	\ и /
0Fh	Tab		2Ch	Z и z	Я и я
10h	Q и q	Й и й	2Dh	X и x	Ч и ч
11h	W и w	Ц и ц	2Eh	C и c	С и с
12h	E и e	У и у	2Fh	V и v	М и м
13h	R и r	К и к	30h	B и b	И и и
14h	T и t	Е и е	31h	N и n	Т и т
15h	Y и y	Н и н	32h	M и m	Ь и ь
16h	U и u	Г и г	33h	, и <	Б и б
17h	I и i	Ш и ш	34h	. и >	Ю и ю
18h	O и o	Щ и щ	35h	/ и ?	. и ,
19h	P и p	З и з	36h	Правая клавиша Shift	
1Ah	[и {	Х и х	37h	*	
1Bh	] и }	Ъ и ъ	38h	Alt	
1Ch	Enter		39h	Пробел	
1Dh	Ctrl		3Ah	Caps Lock	

Таблица 1.3. Скан-коды BIOS для клавиш функциональной, дополнительной и цифровой групп, а также для комбинаций клавиш

Код	Клавиша или комбинация клавиш	Код	Клавиша или комбинация клавиш	Код	Клавиша или комбинация клавиш
3Bh	F1	54h	Shift+F1	6Dh	Alt+F6
3Ch	F2	55h	Shift+F2	6Eh	Alt+F7
3Dh	F3	56h	Shift+F3	6Fh	Alt+F8
3Eh	F4	57h	Shift+F4	70h	Alt+F9
3Fh	F5	58h	Shift+F5	71h	Alt+F10
40h	F6	59h	Shift+F6	72h	Ctrl+Print Screen
41h	F7	5Ah	Shift+F7	73h	Ctrl+←
42h	F8	5Bh	Shift+F8	74h	Ctrl+→
43h	F9	5Ch	Shift+F9	75h	Ctrl+End
44h	F10	5Dh	Shift+F10	76h	Ctrl+Home

продолжение ☞

Таблица 1.3 (*продолжение*)

Код	Клавиша или комбинация клавиш	Код	Клавиша или комбинация клавиш	Код	Клавиша или комбинация клавиш
45h	Num Lock	5Eh	Ctrl+F1	77h	Ctrl+Page Down
46h	Scroll Lock	5Fh	Ctrl+F2	78h	Alt+1
47h	Home	60h	Ctrl+F3	79h	Alt+2
48h	↑	61h	Ctrl+F4	7Ah	Alt+3
49h	Page Up	62h	Ctrl+F5	7Bh	Alt+4
4Ah	-	63h	Ctrl+F6	7Ch	Alt+5
4Bh	←	64h	Ctrl+F7	7Dh	Alt+6
4Ch	5	65h	Ctrl+F8	7Eh	Alt+7
4Dh	→	66h	Ctrl+F9	7Fh	Alt+8
4Eh	+	67h	Ctrl+F10	80h	Alt+9
4Fh	End	68h	Alt+F1	81h	Alt+0
50h	?	69h	Alt+F2	82h	Alt+-
51h	Page Down	6Ah	Alt+F3	83h	Alt+=
52h	Insert	6Bh	Alt+F4	84h	Ctrl+Page Up
53h	Delete	6Ch	Alt+F5		

Ввод информации с клавиатуры при помощи функций BIOS

Клавиатура является основным устройством ввода алфавитно-цифровой информации, а часто – и основным средством управления работой компьютера. Для ввода информации с клавиатуры можно использовать либо функции операционной системы, либо прямой опрос контроллера клавиатуры. Мы не будем рассматривать функции MS-DOS, используемые для ввода данных с клавиатуры, так как они достаточно подробно описаны в литературе [3, 10], но непригодны для сколько-нибудь серьезной работы. Функции DOS имеют два очень серьезных недостатка. Первый недостаток заключается в том, что они не позволяют полностью реализовать возможности функциональных клавиш. Второй недостаток – клавиатурные функции DOS предназначены для работы в режиме терминала (с построчным выводом информации сверху вниз и прокруткой изображения снизу вверх). В процессе считывания символа они выполняют ряд дополнительных операций, что делает весьма неудобным их использование в любом другом, не терминальном режиме.

Функции BIOS обладают гораздо более широкими возможностями, чем функции DOS. Этих возможностей вполне достаточно для выполнения любых операций реальном режиме работы процессора. Вызов клавиатурных функций BIOS выполняется по прерыванию Int 16h. Рассмотрим эти функции.

Прерывание Int 16h, функция 00h: прочитать символ с клавиатуры

Функция 00h считывает символ из буфера клавиатуры и выдает его ASCII-код и скан-код (символ после считывания будет удален из буфера клавиатуры).

Перед вызовом прерывания требуется записать в регистр AH значение 00h.

После выполнения функции в регистры будет помещена следующая информация:

- в AH — скан-код символа;
- в AL — ASCII-код символа.

При нажатии управляющих клавиш функция выдает ASCII-код со значением 0, благодаря чему их легко можно отличить от алфавитно-цифровых. Однако друг от друга управляющие клавиши, таким образом, отличаются только по скан-кодам.

Прерывание Int 16h, функция 01h: получить состояние клавиатуры

Функция 01h проверяет наличие символа в буфере клавиатуры. Если символ присутствует в буфере, функция выдает его ASCII-код и скан-код (не удаляя символ из буфера).

Перед вызовом прерывания требуется записать в регистр AH значение 01h.

После выполнения функции при отсутствии символа в буфере будет установлен флаг ZF. При наличии символа в буфере флаг ZF будет сброшен и в регистры помещена следующая информация:

- в AH — скан-код символа;
- в AL — ASCII-код символа.

Данная функция применяется в тех случаях, когда для управления работой программы используется не только клавиатура, но и другие устройства. Простейший способ работы в этом случае — пооче-

редный циклический опрос всех источников информации. Использовать функцию, аналогичную 00h, нельзя — она заблокирует опрос всех остальных устройств до тех пор, пока не будет введен какой-либо символ с клавиатуры. Поэтому в цикле осуществляется только контроль поступления новой информации, а считывание информации выполняется вне цикла.

Прерывание Int 16h, функция 02h: получить состояние флагов клавиатуры

Функция 02h выдает содержимое байта флагов BIOS.

Перед вызовом прерывания требуется записать в регистр AH значение 02h.

После выполнения функции регистр AL содержит описание состояния флагов:

- бит 0 — правая клавиша Shift (0 — не нажата, 1 — нажата);
- бит 1 — левая клавиша Shift (0 — не нажата, 1 — нажата);
- бит 2 — клавиша Ctrl (0 — не нажата, 1 — нажата);
- бит 3 — клавиша Alt (0 — не нажата, 1 — нажата);
- бит 4 — переключатель Scroll Lock (0 — выключен, 1 — включен);
- бит 5 — переключатель Num Lock (0 — выключен, 1 — включен);
- бит 6 — переключатель Caps Lock (0 — выключен, 1 — включен);
- бит 7 — переключатель Insert (0 — выключен, 1 — включен).

Функция 02h имеет один серьезный недостаток, который сильно ограничивает возможность ее использования совместно с функцией 00h: никак не фиксируется момент изменения состояния управляющих клавиш. Если было выполнено две операции, одна из которых изменила состояние алфавитно-цифровой клавиши, а другая — состояние управляющей клавиши, то далеко не всегда можно определить, какая из операций произошла раньше.

Прерывание Int 16h, функция 03h: управление режимом автоповтора

Функция 03h устанавливает характеристики режима автоповтора.

Перед вызовом прерывания требуется записать в регистры следующую информацию:

- в AH — значение 03h;
- в AL — значение 05h;

- в BH — код, задающий значение задержки автоповтора (табл. 1.4);
- в BL — код, задающий скорость повторения (табл. 1.5).

Таблица 1.4. Коды задержки автоповтора

Код	Задержка, мс
0	250
1	500
2	750
3	1000

Таблица 1.5. Коды частоты автоповтора

Код скорости	Скорость повторения, символ/с	Код скорости	Скорость повторения, символ/с
00h	30,0	10h	7,5
01h	26,7	11h	6,7
02h	24,0	12h	6,0
03h	21,8	13h	5,5
04h	20,0	14h	5,0
05h	18,5	15h	4,6
06h	17,1	16h	4,3
07h	16,0	17h	4,0
08h	15,0	18h	3,7
09h	13,3	19h	3,3
0Ah	12,0	1Ah	3,0
0Bh	10,9	1Bh	2,7
0Ch	10,0	1Ch	2,5
0Dh	9,2	1Dh	2,3
0Eh	8,6	1Eh	2,1
0Fh	8,0	1Fh	2,0

Прерывание Int 16h, функция 04h: включить/выключить звуковой сигнал клавиш

Функция 04h предназначена для включения и выключения звукового сигнала клавиш (щелчка).

Перед вызовом прерывания требуется записать в регистры следующую информацию:

- в AH — значение 04h;

- в AL — код выполняемой операции (0 — отключить сигнал, 1 — включить сигнал).

Прерывание Int 16h, функция 05h: поместить символ в буфер клавиатуры

Функция 05h заносит ASCII-код и скан-код клавиши в буфер, имитируя ввод данных с клавиатуры.

Перед вызовом прерывания требуется записать в регистры следующую информацию:

- в AH — значение 05h;
- в CH — скан-код клавиши;
- в CL — ASCII-код клавиши.

После выполнения функции имеет место следующая ситуация:

- если функция выполнена успешно, флаг переноса сбрасывается, а в регистре AL находится значение 0;
- если функция не выполнена (буфер переполнен), то флаг переноса установлен, а в регистре AL находится значение 1.

Применяется данная функция при отладке программ, осуществляющих ввод информации с клавиатуры, — ее можно использовать для имитации нажатия клавиш.

Прерывание Int 16h, функция 10h: прочитать символ с расширенной клавиатуры

Функция 10h — это усовершенствованный вариант функции 00h. Она позволяет получить скан-коды клавиш F11 и F12, а также клавиш управления курсором.

Перед вызовом прерывания требуется записать в регистр AH значение 10h.

После выполнения функции в регистры будет помещена следующая информация:

- в AH —расширенный скан-код символа;
- в AL — ASCII-код символа.

Функция 10h в качестве признака нажатия управляющей клавиши использует ASCII-коды 0, 0Ah, 0Dh, E0h. Отличия ее от функции 00h иллюстрирует табл. 1.6.

Таблица 1.6. Расширенные скан-коды, возвращаемые функцией 10h для клавиш функциональной, дополнительной и цифровой групп

Скан-код	ASCII-код	Клавиша или комбинация клавиш	Скан-код	ASCII-код	Клавиша или комбинация клавиш
73h	0	Ctrl+4	90h	0	Ctrl++
	E0h	Ctrl+←	91h	0	Ctrl+2
74h	0	Ctrl+6		E0h	Ctrl+?
	E0h	Ctrl+→	92h	0	Ctrl+0
75h	0	Ctrl+1		E0h	Ctrl+Insert
	E0h	Ctrl+End	93h	0	Ctrl+.
76h	0	Ctrl+3		E0h	Ctrl+Delete
	E0h	Ctrl+Page Down	94h	0	Ctrl+Tab
77h	0	Ctrl+7	95h	0	Ctrl+/
	E0h	Ctrl+Home	96h	0	Ctrl+*
84h	0	Ctrl+9	97h	0	Alt+Home
	0Eh	Ctrl+Page Up	98h	0	Alt+↑
85h	0	F11	99h	0	Alt+Page Up
86h	0	F12	9Bh	0	Alt+←
87h	0	Shift+F11	9Dh	0	Alt+→
88h	0	Shift+F12	9Fh	0	Alt+End
89h	0	Ctrl+F11	A0h	0	Alt+?
8Ah	0	Ctrl+F12	A1h	0	Alt+Page Down
8Bh	0	Alt+F11	A2h	0	Alt+Insert
8Ch	0	Alt+F12	A3h	0	Alt+Delete
8Dh	0	Ctrl+8	A5h	0	Alt+Tab
	0Eh	Ctrl+↑	E0h	0Ah	Ctrl+Enter
8Eh	0	Ctrl+-		0Dh	Enter
8Fh	0	Ctrl+5			

Прерывание Int 16h, функция 11h: получить состояние расширенной клавиатуры

Функция 11h — усовершенствованный вариант функции 01h. Данная функция позволяет получить скан-коды клавиш F11 и F12, а также клавиш управления курсором.

Перед вызовом прерывания требуется записать в регистр AH значение 11h.

После выполнения функции при наличии символа в буфере клавиатуры должна иметь место следующая ситуация:

- флаг нуля (ZF) сброшен;
- в AH —расширенный скан-код символа;
- в AL — ASCII-код символа.

При отсутствии символа в буфере флаг нуля будет установлен.

Также как и функция 01h, данная функция только проверяет наличие символа, но не извлекает его, то есть не стирает из буфера.

Прерывание Int 16h, функция 12h: получить состояние флагов расширенной клавиатуры

Функция 12h — это усовершенствованный вариант функции 02h. Она позволяет получить состояние специальных клавиш расширенной клавиатуры.

Функция выдает содержимое байта флагов BIOS.

Перед вызовом прерывания требуется записать в регистр AH значение 12h.

После выполнения функции регистр AX содержит описание состояния флагов:

- бит 0 — правая клавиша Shift (0 — не нажата, 1 — нажата);
- бит 1 — левая клавиша Shift (0 — не нажата, 1 — нажата);
- бит 2 — клавиши Ctrl (0 — не нажаты, 1 — нажата, по крайней мере, одна из клавиш);
- бит 3 — клавиши Alt (0 — не нажаты, 1 — нажата, по крайней мере, одна из клавиш);
- бит 4 — переключатель Scroll Lock (0 — выключен, 1 — включен);
- бит 5 — переключатель Num Lock (0 — выключен, 1 — включен);
- бит 6 — переключатель Caps Lock (0 — выключен, 1 — включен);
- бит 7 — переключатель Insert (0 — выключен, 1 — включен);
- бит 8 — левая клавиша Ctrl (0 — не нажата, 1 — нажата);
- бит 9 — левая клавиша Alt (0 — не нажата, 1 — нажата);
- бит 10 — правая клавиша Ctrl (0 — не нажата, 1 — нажата);

- бит 11 — правая клавиша Alt (0 — не нажата, 1 — нажата);
- бит 12 — клавиша Scroll Lock (0 — не нажата, 1 — нажата);
- бит 13 — клавиша Num Lock (0 — не нажата, 1 — нажата);
- бит 14 — клавиша Caps Lock (0 — не нажата, 1 — нажата);
- бит 15 — клавиша SysReq (0 — не нажата, 1 — нажата).

Функция 12h имеет тот же недостаток, что и функция 02h, — не всегда можно однозначно определить, какая из операций произошла раньше: переключение управляющей или алфавитно-цифровой клавиши.

Примеры использования функций BIOS

Пример, показанный в листинге 1.1, предназначен для демонстрации особенностей функции 00h. Программа TestInt16_00h выполняет одну простую операцию — отображает на экране ASCII-коды и скан-коды клавиш, которые пользователь нажимает на клавиатуре.

Программа функционирует в текстовом видеорежиме, особенности которого мы рассмотрим в главе 4 «Видеоконтроллеры». Вывод информации осуществляется напрямую в видеопамять, которая в цветном текстовом режиме размещается по адресу B8000h. Каждому символу экрана соответствует байт кода (ASCII) и байт атрибута, описывающий цвет символа и цвет фона знакоместа. Видеопамять при выводе на экран разбивается по строкам: в строке 80 символов, то есть 160 байт информации; вывод выполняется справа налево, сверху вниз. Таким образом, байт видеопамяти с нулевым номером соответствует ASCII-коду первого символа, следующий байт определяет его цвет, третий байт соответствует ASCII-коду второго символа и т. д.

Листинг 1.1. Тестирование функции ввода с клавиатуры 00h

```
IDEAL
P386
LOCALS
MODEL MEDIUM

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Счетчик операций нажатия/отпускания клавиш
```

продолжение ⇨

Листинг 1.1 (*продолжение*)

```
PressCounter DW ?
; Текстовые сообщения
Text DB 0,18
     DB "Тестирование функции ввода с клавиатуры 00h",0
     DB 2,0,"ASCII-код  Скан-код",0
     DB 24,29,"Нажмите любую клавишу",0
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC TestInt16_00h
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        mov     CX,3
        mov     SI,offset Text
@@OutText:
        call    ShowString
        loop    @@OutText
; Установить начальную позицию для вывода кодов
; в нулевой колонке пятой строки
        mov     [ScreenString],3
        mov     [ScreenColumn],0
; Инициализировать счетчик операций
; нажатия/отпускания клавиш
        mov     [PressCounter],0

; Принять очередную пару кодов с клавиатуры
@@Next: mov     AH,0
        int     16h
; Отобразить принятые коды в шестнадцатеричном виде
        ; Отобразить ASCII-код (начиная с 3-й колонки)
        mov     [ScreenColumn],3
        call    ShowHexByte
        ; Отобразить скан-код (начиная с 14-й колонки)
        mov     [ScreenColumn],14
        mov     AL,AH
        call    ShowHexByte
        ; Перейти на следующую строку
```

```
        inc     [ScreenString]
        ; Увеличить значение счетчика нажатий клавиш
        inc     [PressCounter]
        ; После 20 нажатий выйти из цикла
        cmp     [PressCounter],20
        jb      @@Next

; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP TestInt16_00h
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"

END
```

В программе TestInt16_00h используется несколько вспомогательных процедур, не относящихся непосредственно к клавиатурным функциям BIOS, но необходимых для отображения информации на экране. Подобные процедуры общего назначения объединены в отдельную группу, оформленную в виде подключаемого файла (include-файла), приведенного в листинге 1.2. В последующих примерах данный файл также будет использоваться, что позволяет избежать многократного дублирования кодов. Назначение подпрограмм, включенных в листинг 1.2, следующее:

- процедура ShowASCIIChar осуществляет вывод символа в ASCII-коде в заданную позицию экрана;
- процедура ShowHexByte отображает в заданную позицию экрана содержимое регистра AL (байт данных) в шестнадцатеричном коде;
- процедура ShowHexWord отображает в заданную позицию экрана содержимое регистра AX (слово данных) в шестнадцатеричном коде;
- процедура ShowHexDWord отображает в заданную позицию экрана содержимое регистра EAX (двойное слово) в шестнадцатеричном коде;
- процедура ShowBinByte отображает в заданную позицию экрана содержимое регистра AL (байт данных) в двоичном коде;
- процедура ShowBinDWord отображает в заданную позицию экрана содержимое регистра EAX (двойное слово) в двоичном коде;

- процедура ShowString выводит текстовую строку в заданную область экрана, причем используется цвет символов и фона, заданный по умолчанию;
- процедура ShowText использует процедуру ShowString для вывода текста (группы строк, цвет которых определяется по умолчанию) на экран;
- процедура ShowColorString выводит текстовую строку заданного цвета в заданную область экрана;
- процедура ShowColorText использует процедуру ShowColorString для вывода разноцветного текста (группы строк, цвет каждой из которых задается индивидуально) на экран;
- процедура ShowASCIIField служит для вывода на экран текстового поля фиксированного размера;
- процедура SetCursorPosition предназначена для управления положением курсора, но в данном примере используется только для того, чтобы убрать курсор за пределы экрана (сделать курсор невидимым);
- процедура GetChar ожидает ввода любого символа с клавиатуры, а затем считывает ASCII-код и скан-код этого символа;
- процедура WaitChar проверяет наличие символа в буфере клавиатуры, и если символ есть — считывает его ASCII-код и скан-код;
- процедура ClearScreen осуществляет очистку экрана;
- процедура Beep предназначена для выдачи оператору звукового сигнала (обычно в случае какой-либо ошибки);
- процедура FatalError осуществляет выдачу сообщения о фатальной ошибке, после чего производится экстренное (аварийное) прекращение работы программы.

Процедуры вывода информации на экран используют метод прямой записи в видеопамять с учетом особенностей текстового режима:

- каждый символ кодируется в видеопамяти двумя байтами: первый байт содержит код символа, а второй описывает цвет символа и фон знакоместа;
- видеопамять отображена на адресное пространство процессора до адреса 0B8000h (для доступа к ней обычно применяются сегментный регистр ES, в который записывается число B800h, и любой индексный регистр, в который заносится смещение символа от начала видеопамяти);

- длина одной строки составляет 80 символов (160 байт);
- на экране помещается 25 текстовых строк.

Листинг 1.2. Процедуры ввода-вывода общего назначения для работы в цветном текстовом режиме

```
DATASEG
; Цвет и фон выводимого текста (по умолчанию установим
; вывод белого текста по черному фону)
TextColorAndBackground DB 0Fh
; Начальная позиция для вывода текстовой строки на экран
ScreenString DW ?
ScreenColumn DW ?
ENDS

CODESEG
; Адрес основного сегмента данных
MainDataSeg DW ?

;*************************************************
;*        ВЫВОД БАЙТА НА ЭКРАН В КОДЕ ASCII       *
;* Подпрограмма выводит содержимое регистра AL в  *
;* коде ASCII в указанную позицию экрана.         *
;* Координаты позиции передаются через глобальные *
;* переменные ScreenString и ScreenColumn. После  *
;* выполнения операции вывода происходит автомати- *
;* ческое приращение значений этих переменных.    *
;*************************************************
PROC ShowASCIIChar near
        pusha
        push    DS
        push    ES
        mov     DI,[CS:MainDataSeg]
        mov     DS,DI
        cld
; Настроить пару ES:DI для прямого вывода в видеопамять
        push    AX
        ; Загрузить адрес сегмента видеоданных в ES
        mov     AX,0B800h
        mov     ES,AX
        ; Умножить номер строки на длину строки в байтах
        mov     AX,[ScreenString]
        mov     DX,160
        mul     DX
        ; Прибавить номер колонки (дважды)
        add     AX,[ScreenColumn]
        add     AX,[ScreenColumn]
        ; Переписать результат в индексный регистр
```

продолжение ⇨

Листинг 1.2 (*продолжение*)

```
        mov     DI,AX
        pop     AX
        mov     AH,[TextColorAndBackground]
        stosw

; Подготовка для вывода следующих байтов
        ; Перевести текущую позицию на 2 символа влево
        inc     [ScreenColumn]
        ; Проверить пересечение правой границы экрана
        cmp     [ScreenColumn],80
        jb      @@End
        ; Если достигнута правая граница экрана -
        ; перейти на следующую строку
        sub     [ScreenColumn],80
        inc     [ScreenString]
@@End:  pop     ES
        pop     DS
        popa
        ret
ENDP ShowASCIIChar

;****************************************************
;*    ВЫВОД БАЙТА НА ЭКРАН В ШЕСТНАДЦАТЕРИЧНОМ КОДЕ    *
;* Подпрограмма выводит содержимое регистра AL        *
;* в шестнадцатеричном коде в заданную позицию экрана. *
;* Координаты позиции передаются через глобальные     *
;* переменные ScreenString и ScreenColumn. После      *
;* выполнения операции вывода происходит автомати-    *
;* ческое приращение значений этих переменных.        *
;****************************************************
PROC ShowHexByte near
        pusha
        push    DS
        push    ES
; Настроить DS на глобальный сегмент данных
        mov     DI,[CS:MainDataSeg]
        mov     DS,DI
        cld

; Настроить пару ES:DI для прямого вывода в видеопамять
        push    AX
        ; Загрузить адрес сегмента видеоданных в ES
        mov     AX,0B800h
        mov     ES,AX
        ; Умножить номер строки на длину строки в байтах
        mov     AX,[ScreenString]
        mov     DX,160
```

```
        mul     DX
        ; Прибавить к полученному произведению номер
        ; колонки (дважды)
        add     AX,[ScreenColumn]
        add     AX,[ScreenColumn]
        ; Переписать результат в индексный регистр
        mov     DI,AX
        pop     AX

; Использовать цвет символов, заданный по умолчанию
        mov     AH,[TextColorAndBackground]
; Вывести старший разряд числа
        push    AX
        ; Выделить старший разряд
        shr     AL,4
        ; Преобразовать старший разряд в код ASCII
        add     AL,'0'
        cmp     AL,'9'
        jbe     @@M0
        add     AL,'A'-'9'- 1
        ; Вывести разряд числа на экран
@@M0:   stosw
        pop     AX
; Вывести младший разряд числа
        ; Выделить младший разряд числа
        and     AL,0FH
        ; Преобразовать младший разряд в код ASCII
        add     AL,'0'
        cmp     AL,'9'
        jbe     @@M1
        add     AL,'A'-'9'- 1
        ; Вывести разряд числа на экран
@@M1:   stosw

; Подготовка для вывода следующих байтов
        ; Перевести текущую позицию на 2 символа влево
        add     [ScreenColumn],2
        ; Проверить пересечение правой границы экрана
        cmp     [ScreenColumn],80
        jb      @@End
        ; Если достигнута правая граница экрана
        ; перейти на следующую строку
        sub     [ScreenColumn],80
        inc     [ScreenString]
@@End:  pop     ES
        pop     DS
        popa
        ret
ENDP ShowHexByte
```

продолжение ⇨

Листинг 1.2 (*продолжение*)

```
;****************************************************
;*         ВЫВОД 16-РАЗРЯДНОГО СЛОВА НА ЭКРАН       *
;*              В ШЕСТНАДЦАТЕРИЧНОМ КОДЕ            *
;* Параметры:                                       *
;* AX - число, которое будет выведено на экран.     *
;* Номер строки передается через глобальную         *
;* переменную ScreenString, номер столбца - через   *
;* переменную ScreenColumn,.цвет текста определяется *
;* глобальной переменой TextColorAndBackground.     *
;****************************************************
PROC ShowHexWord NEAR
        xchg    AL,AH
        call    ShowHexByte
        xchg    AL,AH
        call    ShowHexByte
        ret
ENDP ShowHexWord

;****************************************************
;*         ВЫВОД 32-РАЗРЯДНОГО СЛОВА НА ЭКРАН       *
;*              В ШЕСТНАДЦАТЕРИЧНОМ КОДЕ            *
;* Параметры:                                       *
;* EAX - число, которое будет выведено на экран.    *
;* Номер строки передается через глобальную         *
;* переменную ScreenString, номер столбца - через   *
;* переменную ScreenColumn, цвет текста определяется *
;* глобальной переменой TextColorAndBackground.     *
;****************************************************
PROC ShowHexDWord NEAR
        rol     EAX,8
        call    ShowHexByte
        rol     EAX,8
        call    ShowHexByte
        rol     EAX,8
        call    ShowHexByte
        rol     EAX,8
        call    ShowHexByte
        ret
ENDP ShowHexDWord

;************************************************
;*      ВЫВОД БАЙТА НА ЭКРАН В ДВОИЧНОМ КОДЕ    *
;* Подпрограмма выводит содержимое регистра AL  *
;* в двоичном коде в указанную позицию экрана.  *
;* Координаты позиции передаются через глобальные *
;* переменные ScreenString и ScreenColumn. После *
;* выполнения операции вывода происходит автома- *
;* тическое приращение значений этих переменных. *
;************************************************
```

```
PROC ShowBinByte near
        pusha
        push    DS
        push    ES
        ; Копируем отображаемый байт в BL
        mov     BL,AL
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
        cld
        ; Загрузить адрес "текстовой" видеопамяти в ES
        mov     AX,0BB00h
        mov     ES,AX
        ; Умножить номер строки на длину строки в байтах
        mov     AX,[ScreenString]
        mov     DX,160
        mul     DX
        ; Прибавить дважды номер колонки
        add     AX,[ScreenColumn]
        add     AX,[ScreenColumn]
        ; Переписать результат в индексный регистр
        mov     DI,AX

; Отобразить разряды числа (начиная со старшего)
        mov     AH,[TextColorAndBackground]
        mov     CX,B    ;счетчик разрядов
@@L0:   mov     AL,'0'
        ; Выделить очередной разряд числа
        rol     BL,1
        jnc     @@L1
        mov     AL,'1'
        ; Вывести разряд числа на экран
@@L1:   stosw
        loop    @@L0

; Подготовка для вывода следующих байтов
        ; Перевести текущую позицию на 8 символов влево
        add     [ScreenColumn],8
        ; Проверить пересечение правой границы экрана
        cmp     [ScreenColumn],80
        jb      @@End
        ; Если достигнута правая граница экрана -
        ; перейти на следующую строку
        sub     [ScreenColumn],B0
        inc     [ScreenString]
; Конец подпрограммы
@@End:  pop     ES
        pop     DS
        popa
        ret
ENDP ShowBinByte
```

продолжение

Листинг 1.2 (*продолжение*)

```
;****************************************************
;* ВЫВОД 16-РАЗРЯДНОГО СЛОВА НА ЭКРАН В ДВОИЧНОМ КОДЕ *
;* Параметры:                                        *
;* AX - число, которое будет выведено на экран.      *
;* Номер строки передается через глобальную          *
;* переменную ScreenString, номер столбца - через    *
;* переменную ScreenColumn, цвет текста определяется *
;* глобальной переменой TextColorAndBackground.      *
;****************************************************
PROC ShowBinWord NEAR
        rol     AX,8
        call    ShowBinByte
        inc     [ScreenColumn]
        rol     AX,8
        call    ShowBinByte
        ret
ENDP ShowBinWord

;****************************************************
;* ВЫВОД 32-РАЗРЯДНОГО СЛОВА НА ЭКРАН В ДВОИЧНОМ КОДЕ *
;* Параметры:                                        *
;* EAX - число, которое будет выведено на экран.     *
;* Номер строки передается через глобальную          *
;* переменную ScreenString, номер столбца - через    *
;* переменную ScreenColumn, цвет текста определяется *
;* глобальной переменой TextColorAndBackground.      *
;****************************************************
PROC ShowBinDWord NEAR
        rol     EAX,8
        call    ShowBinByte
        inc     [ScreenColumn]
        rol     EAX,8
        call    ShowBinByte
        inc     [ScreenColumn]
        rol     EAX,8
        call    ShowBinByte
        inc     [ScreenColumn]
        rol     EAX,8
        call    ShowBinByte
        ret
ENDP ShowBinDWord

;**************************************************
;*          ВЫВОД ТЕКСТОВОЙ СТРОКИ НА ЭКРАН        *
;* Все параметры передаются через одну структуру:  *
;* первый байт - номер начальной строки (0-24);    *
;* второй байт - номер начальной колонки (0-79);   *
;* далее идет строка, ограниченная нулем.          *
;* Адрес структуры передается через регистры DS:SI. *
;**************************************************
```

```
PROC ShowString near
        push    AX
        push    BX
        push    DI
        push    ES
; Настроить регистр ES на глобальный сегмент данных
        mov     AX,[CS:MainDataSeg]
        mov     ES,AX
; Запомнить цвет текста в BL
        mov     BL,[ES:TextColorAndBackground]
; Настроить регистр ES на видеопамять
        mov     AX,0B800h
        mov     ES,AX
        cld
; Вычислить адрес для строки в видеопамяти
        ; Загрузить номер строки экрана в AL и
        ; умножить его на длину строки в байтах
        lodsb
        ; Проверка: номер строки не должен превышать
        ; предела нижней границы экрана
        cmp     AL,24
        ja      @@Error
        mov     AH,160
        mul     AH
        ; Переписать результат в индексный регистр DI
        mov     DI,AX
        ; Загрузить номер столбца и дважды
        ; прибавить его к DI
        lodsb
        cmp     AL,79   ;номер колонки не должен
        ja      @@Error ;превышать ширины экрана
        mov     BH,AL   ;запомнить номер колонки
        xor     AH,AH   ;обнулить AH
        add     DI,AX
        add     DI,AX
        ; Загрузить атрибут цвета в AH
        mov     AH,BL
@@L1:   ; Загрузить очередной символ строки в AL
        lodsb
        ; Проверка на 0 (на конец строки)
        and     AL,AL
        jz      @@L2
        ; Проверить номер колонки символа
        cmp     BH,79
        ja      @@Error ;нарушена правая граница экрана
        ; Вывести символ на экран
        stosw
        inc     BH      ;увеличить номер колонки
        jmp     @@L1
```

продолжение ⇗

Листинг 1.2 (*продолжение*)

```
@@L2:   pop     ES
        pop     DI
        pop     BX
        pop     AX
        ret
@@Error: ;Немедленный выход в DOS по ошибке
        mov     AH,4Ch
        int     21h
ENDP ShowString

;****************************************************
;*          ВЫВОД ТЕКСТА (ГРУППЫ СТРОК) НА ЭКРАН     *
;* ShowText использует процедуру ShowString для вывода *
;* на экран группы строк.                            *
;* Параметры:                                        *
;* CX - количество строк;                            *
;* DS:SI - адрес первой строки в группе.             *
;* Строки должны иметь заданный для ShowString формат *
;* и располагаться в памяти последовательно.         *
;* При выводе текста используются принятые по        *
;* умолчанию цвет и фон.                             *
;****************************************************
PROC ShowText near
; Цикл вывода строк
@@NextString:
        call    ShowString
        loop    @@NextString
; Процедура не сохраняет значения в CX и SI
        ret
ENDP ShowText

;***************************************************
;*  ВЫВОД ТЕКСТОВОЙ СТРОКИ ЗАДАННОГО ЦВЕТА НА ЭКРАН  *
;* Все параметры передаются через одну структуру:   *
;* первый байт - атрибут цвета и фона для строки;   *
;* второй байт - номер начальной строки (0-24);    *
;* третий байт - номер начальной колонки (0-79);    *
;* далее идет строка, ограниченная нулем.           *
;* Адрес структуры передается через регистры DS:SI.  *
;***************************************************
PROC ShowColorString near
        push    AX
        ; Запомнить цвет, используемый по уиолчанию
        mov     AL,[TextColorAndBackground]
        push    AX
        ; Установить цвет строки
        cld
        lodsb
```

```
        mov     [TextColorAndBackground],AL
        ; Использовать функцию ShowString
        call    ShowString
        ; Восстановить цвет, используемый по умолчанию
        pop     AX
        mov     [TextColorAndBackground],AL
        pop     AX
        ret
ENDP ShowColorString

;*****************************************************
;*    ВЫВОД ЦВЕТНОГО ТЕКСТА (ГРУППЫ СТРОК) НА ЭКРАН    *
;* ShowColorText использует процедуру ShowColorString  *
;* для вывода на экран группы разноцветных строк.      *
;* Параметры:                                          *
;* CX - количество строк;                              *
;* DS:SI - адрес первой строки в группе.               *
;* Строки должны иметь заданный для ShowColorString    *
;* формат и располагаться в памяти последовательно.    *
;* При выводе текста используются принятые по          *
;* умолчанию цвет и фон.                               *
;*****************************************************
PROC ShowColorText near
; Цикл вывода строк
@@NextColorString:
        call    ShowColorString
        loop    @@NextColorString
; Процедура не сохраняет значения в CX и SI
        ret
ENDP ShowColorText

;*******************************
;*  УСТАНОВИТЬ ПОЗИЦИЮ КУРСОРА  *
;* Входные параметры:           *
;* ScreenString - номер строки  *
;* ScreenColumn - номер столбца *
;*******************************
PROC SetCursorPosition NEAR
        pusha
; Вычисление линейного адреса курсора
        mov     AX,[ScreenString]
        mov     BX,80
        mul     BX
        add     AX,[ScreenColumn]
        mov     BL,AL ;запомнить младший байт
; Прямой вывод позиции курсора
; в регистры видеоконтроллера
        mov     DX,3D4h
        ; Вывести старший байт адреса курсора
```

продолжение ⇗

Листинг 1.2 (*продолжение*)

```
        mov     AL,0Eh
        out     DX,AX
        ; Вывести младший байт адреса курсора
        inc     AL
        mov     AH,BL
        out     DX,AX
        popa
        ret
ENDP SetCursorPosition

;*********************************************
;*        ПРИНЯТЬ СИМВОЛ ОТ КЛАВИАТУРЫ        *
;* Процедура осуществляет ввод символа с      *
;* помощью функции 00h прерывания Int16h.     *
;* Для "текстовых" управляющих клавиш вместо *
;* скан-кодов используются ASCII-коды.        *
;* Входных параметров нет.                    *
;* Функция возвращает:                        *
;* AL - код символа;                          *
;* AH - управляющий код, если в AL ноль.      *
;*********************************************
PROC GetChar NEAR
; Очистить буфер клавиатуры
@@ClearBuffer:
        mov     AH,1
        int     16h
        jz      @@WaitChar
        mov     AH,0
        int     16h
        jmp short @@ClearBuffer
; Ожидать нажатия клавиши и принять код символа
@@WaitChar:
        mov     AH,0
        int     16h
; Обработать принятый код
        and     AL,AL
        jnz     @@Get1
        ret     ;(в AL - ноль, в AH - управляющий код)
@@Get1: cmp     AL,32
        jnb     @@Get2
        ; Переписать в AH управляющий код
        xchg    AL,AH
        mov     AL,0
        ret     ;(в AL - ноль, в AH - управляющий код)
@@Get2: mov     AH,0
        ret     ;(в AL - код буквы, в AH - ноль)
ENDP GetChar
```

```
;**********************************************
;*       ПРИНЯТЬ СИМВОЛ ОТ КЛАВИАТУРЫ,        *
,*           ЕСЛИ ОН ЕСТЬ В БУФЕРЕ            *
;* Процедура проверяет наличие символа в буфере *
;* клавиатуры и считывает его, если он есть   *
;* Входных параметров нет                     *
;* Функция возвращает                         *
;* AL - код символа,                          *
;* AH  управляющий код, если в AL ноль        *
;* Если в AL и AH нули  нажатий не было       *
;**********************************************
PROC WaitChar NEAR
; Проверить наличие символа в буфере клавиатуры
        mov     AH,1
        int     16h
        jz      @@NoInput
; Принять символ от клавиатуры
        mov     AX,0
        int     16h
        and     AL,AL
        jnz     @@GET1
        ret     ;в AL - ноль, в AH - управляющий код
@@GET1. cmp     AL,32
        jnb     @@GET2
        mov     AH,AL ,переписать в AH управляющий код
        mov     AL,0
        ret     ,в AL - ноль, в AH - управляющий код
@@GET2  mov     AH,0
        ret     ;в AL - код буквы, в AH - ноль
@@NoInput:
        xor     AX,AX
        ret     ,в AL и AH - нули
ENDP WaitChar

;**********************************
;* ОЧИСТКА ЭКРАНА В ТЕКСТОВОМ РЕЖИМЕ *
,*  (процедура параметров не имеет)  *
;**********************************
PROC ClearScreen NEAR
        pusha
        push    ES
; Настроить ES DI на "текстовую" область видеопамяти
        mov     AX,0B800h
        mov     ES,AX
        cld
        mov     DI,0
; Вывести 2000 "пустых" символов (ASCII код 0) с
; атрибутом "белый цвет, черный фон'
        mov     CX,2000
```

продолжение ⇨

Листинг 1.2 (*продолжение*)

```
        mov     AX,0F00H
        rep     stosw
        pop     ES
        popa
        ret
ENDP ClearScreen

;****************************************************
;* ПОДАЧА ЗВУКОВОГО СИГНАЛА ЧЕРЕЗ ВСТРОЕННЫЙ ДИНАМИК *
;*          (процедура параметров не имеет)          *
;****************************************************
PROC Beep NEAR
        push    AX
        push    DX
; Послать на терминал код "звонок" (07h)
        mov     AH,2
        mov     DL,7
        int     21h
        pop     DX
        pop     AX
        ret
ENDP Beep

;***************************************************
;*      ВЫВОД НА ЭКРАН ТЕКСТОВОГО ПОЛЯ ДАННЫХ       *
;* Передаваемые параметры:                          *
;* DS:SI - указатель на структуру данных;           *
;* BX - смещение поля от начала структуры;          *
;* CX - длина поля в байтах.                        *
;* Цвет задается переменной TextColorAndBackground. *
;* Координаты позиции передаются через глобальные   *
;* переменные ScreenString и ScreenColumn.          *
;***************************************************
PROC ShowASCIIField near
        pusha
        push    ES
        mov     AX,0B800h    ;Настроить ES для прямого
        mov     ES,AX        ;вывода на экран
; Установить указатель на начало поля
        add     SI,BX
; Вычислить начальную позицию в видеопамяти
        mov     AX,[ScreenString]
        mov     DI,160
        mul     DI
        add     AX,[ScreenColumn]
        add     AX,[ScreenColumn]
        mov     DI,AX
; Использовать цвет, заданный по умолчанию
        mov     AH,[TextColorAndBackground]
```

```
; Вывести поле на экран
@@NextChar:
        lodsb
        stosw
        loop    @@NextChar
        pop     ES
        popa
        ret
ENDP ShowASCIIField

;****************************************************
;*       ВЫДАЧА СООБЩЕНИЯ О ФАТАЛЬНОЙ ОШИБКЕ        *
;*          И ЭКСТРЕННЫЙ ВЫХОД ИЗ ПРОГРАММЫ          *
;* Параметры:                                       *
;* DS:SI - указатель на строку сообщения об ошибке, *
;* представленную в формате ShowColorString.        *
;****************************************************
PROC FatalError near
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Настроить DS на глобальный сегмент данных
        mov     DI,[CS:MainDataSeg]
        mov     DS,DI
; Вывести сообщение об ошибке красным цветом
        mov     [TextColorAndBackground],12
        call    ShowString
; Переместить курсор в нижнюю часть экрана
        mov     [ScreenString],24
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Аварийный выход из программы
        mov     AH,4Ch
        int     21h
ENDP FatalError
ENDS
```

Приведенная в листинге 1.2 процедура ввода символа GetChar выполняет определенные преобразования над данными, выдаваемыми функцией 00h по прерыванию Int 16h. Дело в том, что некоторые из управляющих символов, перечисленных в табл. 1.1, традиционно имеют как ASCII-коды, так и скан-коды, причем значения этих кодов не совпадают. С целью упрощения последующего анализа кодов введенных символов процедура GetChar переносит ASCII-коды символов со значениями от 0 до 20h из регистра AL в регистр AH, заменяя соответствующие скан-коды (AL при этом обнуляется).

В листинге 1.3. даны мнемонические обозначения для кодов наиболее часто применяемых управляющих клавиш, которые процедура GetChar сохраняет в регистре AH (символ является управляющим,

если в регистре AL был возвращен код 0). Кроме того, в листинг 1.3 включены мнемонические обозначения цветовых оттенков, которые будут применяться во всех последующих программах, работающих в 16-цветных и 256-цветных режимах

Листинг 1.3. Мнемонические обозначения кодов управляющих клавиш и цветовых оттенков

```
; КОДЫ УПРАВЛЯЮЩИХ КЛАВИШ
; Для "текстовых" управляющих клавиш вместо скан-кодов
; используются ASCII-коды:
B_RUBOUT equ   8 ;забой
B_TAB    equ   9 ;табуляция
B_LF     equ  10 ;перевод строки
B_ENTER  equ  13 ;возврат каретки
B_ESC    equ  27 ;"Esc"
; Скан-коды функциональных клавиш:
F1       equ  59
F2       equ  60
F3       equ  61
F4       equ  62
F5       equ  63
F6       equ  64
F7       equ  65
F8       equ  66
F9       equ  67
F10      equ  68
; Скан-коды клавиш дополнительной клавиатуры:
B_HOME   equ  71 ;перейти в начало
B_UP     equ  72 ;стрелка вверх
B_PGUP   equ  73 ;на страницу вверх
B_BS     equ  75 ;стрелка влево
B_FWD    equ  77 ;стрелка вправо
B_END    equ  79 ;перейти в конец
B_DN     equ  80 ;стрелка вниз
B_PGDN   equ  81 ;на страницу вниз
B_INS    equ  82 ;переключить режим (вставка/замещение)
B_DEL    equ  83 ;удалить символ над курсором

; МНЕМОНИЧЕСКИЕ ОБОЗНАЧЕНИЯ ЦВЕТОВ
; "Темные" цвета (можно использовать для фона и текста)
BLACK      equ  0 ;черный
BLUE       equ  1 ;темно-синий
GREEN      equ  2 ;темно-зеленый
CYAN       equ  3 ;бирюзовый (циан)
RED        equ  4 ;темно-красный
MAGENTA    equ  5 ;темно-фиолетовый
BROWN      equ  6 ;коричневый
LIGHTGREY  equ  7 ;серый
; "Светлые" цвета (только для текста)
```

```
DARKGREY     equ  8 ;темно-серый
LIGHTBLUE    equ  9 ;синий
LIGHTGREEN   equ 10 ;зеленый
LIGHTCYAN    equ 11 ;голубой
LIGHTRED     equ 12 ;красный
LIGHTMAGENTA equ 13 ;фиолетовый
YELLOW       equ 14 ;желтый
WHITE        equ 15 ;белый
```

Листинг 1.4 содержит набор макрокоманд, делающих более наглядными и компактными участки программного кода, в которых осуществляются операции ввода-вывода.

Листинг 1.4. Макрокоманды для вывода данных на экран

```
; МАКРОКОМАНДЫ ТЕКСТОВОГО РЕЖИМА
;***********************************
;*    ВЫВОД БАЙТА В ДВОИЧНОМ КОДЕ    *
;* Параметры:                      *
;* SString - номер строки экрана;    *
;* SColumn - номер колонки экрана;   *
;* BData - отображаемый байт данных. *
;***********************************
MACRO MShowBinByte SString,SColumn,BData
        mov     [ScreenString],SString
        mov     [ScreenColumn],SColumn
        mov     AL,BData
        call    ShowBinByte
ENDM

;*******************************************
;* ВЫВОД 16-РАЗРЯДНОГО СЛОВА В ДВОИЧНОМ КОДЕ *
;* Параметры:                              *
;* SString - номер строки экрана;            *
;* SColumn - номер колонки экрана;           *
;* WData - отображаемое слово данных.        *
;*******************************************
MACRO MShowBinWord SString,SColumn,WData
        mov     [ScreenString],SString
        mov     [ScreenColumn],SColumn
        mov     AX,WData
        call    ShowBinWord
ENDM

;*******************************************
;* ВЫВОД 32-РАЗРЯДНОГО СЛОВА В ДВОИЧНОМ КОДЕ *
;* Параметры:                              *
;* SString - номер строки экрана;            *
;* SColumn - номер колонки экрана;           *
;* DData - отображаемое слово данных.        *
;*******************************************
```

продолжение ⇗

Листинг 1.4 (*продолжение*)

```
MACRO MShowBinDWord SString,SColumn,DData
        mov     [ScreenString],SString
        mov     [ScreenColumn],SColumn
        mov     EAX,DData
        call    ShowBinDWord
ENDM

;****************************************
;* ВЫВОД БАЙТА В ШЕСТНАДЦАТЕРИЧНОМ КОДЕ *
;* Параметры:                           *
;* SString - номер строки экрана;       *
;* SColumn - номер колонки экрана;      *
;* BData - отображаемый байт данных.    *
;****************************************
MACRO MShowHexByte SString,SColumn,BData
        mov     [ScreenString],SString
        mov     [ScreenColumn],SColumn
        mov     AL,BData
        call    ShowHexByte
ENDM

;********************************************************
;* ВЫВОД 16-РАЗРЯДНОГО СЛОВА В ШЕСТНАДЦАТЕРИЧНОМ КОДЕ *
;* Параметры:                                           *
;* SString - номер строки экрана;                       *
;* SColumn - номер колонки экрана;                      *
;* WData - отображаемое слово данных.                   *
;********************************************************
MACRO MShowHexWord SString,SColumn,WData
        mov     [ScreenString],SString
        mov     [ScreenColumn],SColumn
        mov     AX,WData
        call    ShowHexWord
ENDM

;********************************************************
;* ВЫВОД 32-РАЗРЯДНОГО СЛОВА В ШЕСТНАДЦАТЕРИЧНОМ КОДЕ *
;* Параметры:                                           *
;* SString - номер строки экрана;                       *
;* SColumn - номер колонки экрана;                      *
;* DData - отображаемое слово данных.                   *
;********************************************************
MACRO MShowHexDWord SString,SColumn,DData
        mov     [ScreenString],SString
        mov     [ScreenColumn],SColumn
        mov     EAX,DData
        call    ShowHexDWord
ENDM
```

```
;**********************************
;*   ВЫВОД БАЙТА В ДЕСЯТИЧНОМ КОДЕ  *
;* Параметры:                       *
;* SString - номер строки экрана;   *
;* SColumn - номер колонки экрана;  *
;* BData - отображаемый байт данных. *
;**********************************
MACRO MShowDecByte SString,SColumn,BData
        mov     [ScreenString],SString
        mov     [ScreenColumn],SColumn
        mov     AL,BData
        call    ShowDecByte
ENDM

;*********************************************
;* ВЫВОД 16-РАЗРЯДНОГО СЛОВА В ДЕСЯТИЧНОМ КОДЕ *
;* Параметры:                                  *
;* SString - номер строки экрана;              *
;* SColumn - номер колонки экрана;             *
;* WData - отображаемое слово данных.          *
;*********************************************
MACRO MShowDecWord SString,SColumn,WData
        mov     [ScreenString],SString
        mov     [ScreenColumn],SColumn
        mov     AX,WData
        call    ShowDecWord
ENDM

;*********************************************
;* ВЫВОД 32-РАЗРЯДНОГО СЛОВА В ДЕСЯТИЧНОМ КОДЕ *
;* Параметры:                                  *
;* SString - номер строки экрана;              *
;* SColumn - номер колонки экрана;             *
;* DData - отображаемое слово данных.          *
;*********************************************
MACRO MShowDecDWord SString,SColumn,DData
        mov     [ScreenString],SString
        mov     [ScreenColumn],SColumn
        mov     EAX,DData
        call    ShowDecDWord
ENDM

;********************************************
;*        ВЫВОД ТЕКСТОВОГО ПОЛЯ ДАННЫХ       *
;* Параметры:                                *
;* SString - номер строки экрана;            *
;* SColumn - номер колонки экрана;           *
;* OOffs - смещение поля от начала структуры; *
;* DSize - длина поля в байтах;              *
;* DS:SI - указатель на структуру данных.    *
;********************************************
```

продолжение

Листинг 1.4 (*продолжение*)

```
MACRO MShowASCIIField SString,SColumn,DOffs,DSize
        mov     [ScreenString],SString
        mov     [ScreenColumn],SColumn
        mov     BX,DOffs
        mov     CX,DSize
        call    ShowASCIIField
ENDM

;*******************************
;*   ВЫВОД ТЕКСТОВОЙ СТРОКИ    *
;* Параметры:                  *
;* TOffset - смещение строки.  *
;*******************************
MACRO MShowString TOffset
        mov     SI,offset TOffset
        call    ShowString
ENDM

;**********************************
;* ВЫВОД ЦВЕТНОЙ ТЕКСТОВОЙ СТРОКИ *
;* Параметры:                     *
;* TOffset - смещение строки.     *
;**********************************
MACRO MShowColorString TOffset
        mov     SI,offset TOffset
        call    ShowColorString
ENDM

;*******************************************
;*         ВЫВОД ОДНОЦВЕТНОГО ТЕКСТА       *
;* Параметры:                              *
;* TStrings - количество строк в тексте;   *
;* TOffset - смещение первой строки текста. *
;*******************************************
MACRO MShowText TStrings,TOffset
        mov     CX,TStrings
        mov     SI,offset TOffset
        call    ShowText
ENDM

;*******************************************
;*         ВЫВОД РАЗНОЦВЕТНОГО ТЕКСТА      *
;* Параметры:                              *
;* TStrings - количество строк в тексте;   *
;* TOffset - смещение первой строки текста. *
;*******************************************
MACRO MShowColorText TStrings,TOffset
        mov     CX,TStrings
        mov     SI,offset TOffset
```

```
        call    ShowColorText
ENDM

;****************************************
;*       ВЫВОД СООБЩЕНИЯ ОБ ОШИБКЕ      *
;* Параметры:                           *
;* TOffset - смещение строки сообщения. *
;****************************************
MACRO MFatalError TOffset
        mov     SI,offset TOffset
        call    FatalError
ENDM

; МАКРОКОМАНДЫ ГРАФИЧЕСКОГО РЕЖИМА
;*****************************
;*   ВЫВОД ТЕКСТОВОЙ СТРОКИ  *
;* Параметры:                *
;* TOffset - смещение строки. *
;*****************************
MACRO MGShowString TOffset
        mov     SI,offset TOffset
        call    GShowString
ENDM

;*********************************************
;*         ВЫВОД ОДНОЦВЕТНОГО ТЕКСТА         *
;* Параметры:                                *
;* TStrings - количество строк в тексте;     *
;* TOffset - смещение первой строки текста.  *
;*********************************************
MACRO MGShowText TStrings,TOffset
        mov     CX,TStrings
        mov     SI,offset TOffset
        call    GShowText
ENDM
```

Программа TestInt16_10h, показанная в листинге 1.5, демонстрирует особенности функции 10h. Основное отличие от программы в листинге 1.1 — в другом номере функции. В программе используются те же самые вспомогательные процедуры из листинга 1.2, но текстовые сообщения окрашены в разные цвета. Выделение цветом — простой и удобный прием, позволяющий оператору быстрее отыскивать на экране нужную информацию. Кроме того, различная окраска изображения позволяет избежать недоразумений при поочередном выполнении программ lst_1_01 и lst_1_03, поскольку текст сообщений в этих процедурах практически совпадает (поочередное выполнение программ lst_1_01 и lst_1_03 необходимо для сравнения между собой функций 00h и 10h).

Листинг 1.5. Тестирование функции ввода с клавиатуры 10h

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл иакросов
include "list1_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Счетчик операций нажатия/отпускания клавиш
PressCounter DW ?
; Текстовые сообщения
Text DB LIGHTMAGENTA,0,1B
     DB "Тестирование функции ввода с клавиатуры 10h",0
     DB YELLOW,2,0,"ASCII-код  Скан-код",0
     DB LIGHTRED,24,29,"Нажиите любую клавишу",0
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC TestInt16_10h
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 3,Text
; Установить белый цвет символов и черный фон
        mov     [TextColorAndBackground],WHITE
; Установить начальную позициш для вывода кодов
; в нулевой колонке пятой строки
        mov     [ScreenString],3
        mov     [ScreenColumn],0
```

```
; Инициализировать счетчик операций
; нажатия/отпускания клавиш
        mov     [PressCounter],0

; Принять очередную пару кодов с клавиатуры
@@Next: mov     AH,10h
        int     16h
; Отобразить принятые коды в шестнадцатеричном виде
        ; Отобразить ASCII-код
        mov     [ScreenColumn],3
        call    ShowHexByte
        ; Отобразить скан-код
        mov     [ScreenColumn],14
        mov     AL,AH
        call    ShowHexByte
        ; Перейти на следующую строку
        inc     [ScreenString]
        ; Увеличить значение счетчика нажатий клавиm
        inc     [PressCounter]
        ; После 20 нажатий выйти из цикла
        cmp     [PressCounter],20
        jb      @@Next

; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP TestInt16_10h
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"

END
```

Контроллер прерываний

Устройства ввода информации сообщают центральному процессору о поступлении новых данных с помощью сигналов прерываний. Прерывания от клавиатуры, мыши PS/2-типа и других периферийных устройств, прежде чем поступить в процессор, проходят через контроллер прерываний, где подвергаются предварительной обработке. Контроллер позволяет управлять приоритетами, прохождением сигналов и адресами векторов прерываний.

Контроллер прерываний IBM AT состоял из двух микросхем Intel 8259, включенных в режиме каскадирования (рис. 1.4). Первая микросхема была ведущей, а вторая — ведомой (ведомый контроллер подключен к входу IRQ2 ведущего). На входы IRQ0, IRQ1, IRQ3–IRQ7 ведущей микросхемы и на входы IRQ8–IRQ15 ведомой поступают запросы прерываний, из которых выбирается немаскированный запрос с наивысшим приоритетом, после чего контроллер вырабатывает сигнал INT и передает в процессор вектор прерывания.

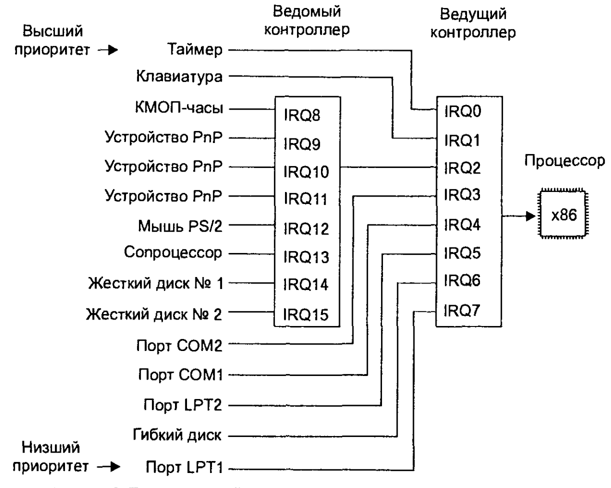


Рис. 1.4. Традиционный порядок подключения внешних устройств к контроллеру прерываний

Вектор формируется путем сложения базового значения (записанного в соответствующий регистр микросхемы) и номера линии, на которую поступил запрос (ведущей микросхеме IRQ0 соответствует линия 0, IRQ7 — линия номер 7; ведомой микросхеме IRQ8 соответствует линия 0, IRQ15 — линия 7). Базовый вектор ведущей микросхемы в реальном режиме DOS имеет значение 08h, базовый вектор ведомой — 70h. Соответственно, ведущая схема вырабатывает вектора с номерами 08h–0Fh, ведомая — с номерами 70h–77h

(табл. 1.7). Приоритеты запросов прерывания (по убыванию) располагаются в следующем порядке: IRQ0, IRQ1, IRQ8–IRQ15, IRQ3–IRQ7.

Таблица 1.7. Аппаратные прерывания AT-совместимых компьютеров

Прерывание	Номер вектора	Адрес вектора	Источник сигнала прерывания
IRQ0	08h	0000:0020h	Системный таймер
IRQ1	09h	0000:0024h	Клавиатура
IRQ2	0Ah	0000:0028h	Ведомая микросхема контроллера
IRQ3	0Bh	0000:002Ch	Последовательный порт COM2
IRQ4	0Ch	0000:0030h	Последовательный порт COM1
IRQ5	0Dh	0000:0034h	Параллельный порт LPT2
IRQ6	0Eh	0000:0038h	Контроллер дисководов гибких дисков
IRQ7	0Fh	0000:003Ch	Параллельный порт LPT1
IRQ8	70h	0000:01C0h	Часы реального времени
IRQ9	71h	0000:01C4h	Любое устройство PnP
IRQ10	72h	0000:01C8h	Любое устройство PnP
IRQ11	73h	0000:01CCh	Любое устройство PnP
IRQ12	74h	0000:01D0h	Мышь PS/2-типа
IRQ13	75h	0000:01D4h	Математический сопроцессор
IRQ14	76h	0000:01D8h	Контроллер жесткого диска № 1
IRQ15	77h	0000:01DCh	Контроллер жесткого диска № 2

Все последующие модели AT-совместимых компьютеров вынуждены имитировать работу микросхем i8259 с целью сохранения совместимости со старым программным обеспечением. Большая часть возможностей указанных микросхем, к счастью для программистов, не используется в AT-совместимых персональных компьютерах: приоритеты прерываний, поступающих от периферийных устройств, и адреса соответствующих векторов жестко зафиксированы (устанавливаются BIOS в процессе начальной загрузки). Полностью перепрограммировать контроллер приходится только при переключении процессора в защищенный режим, так как при этом необходимо изменить номера векторов прерываний, а работа с контроллером в обычном реальном режиме DOS требует выполнения только двух типов операций:

- маскирование и размаскирование прерываний по отдельным линиям;

- посылка сигнала завершения обработки прерывания контроллеру.

Чтобы маскировать (запретить) прерывание, необходимо установить в 1 соответствующий ему бит в регистре маски (номера разрядов маски соответствуют номерам линий сигналов прерывания). Поскольку в маске должен быть изменен только один разряд, а остальные нужно сохранить в исходном состоянии, то вначале требуется прочитать содержимое регистра маски, выполнить операцию изменения соответствующего разряда, а затем записать полученное значение обратно в регистр маски. Между двумя последовательными обращениями к одному и тому же порту рекомендуется вставлять циклы задержки, поскольку контроллер по сравнению с процессором работает слишком медленно. Регистр маски прерываний ведущего контроллера доступен для записи и считывания через порт 21h, ведомого контроллера — через порт A1h.

Например, чтобы запретить прохождение сигнала прерывания от клавиатуры IRQ1 (ведущий контроллер, линия № 1), нужно выполнить следующий ряд команд:

```
; Прочитать регистр маски ведущего контроллера
        in      AL,21h
; Установить второй разряд маски
        or      AL,10b
; Вставить задержку
        jmp short $+2
        jmp short $+2
; Записать маску обратно в регистр
        out     21h,AL
```

Когда программист устанавливает собственный обработчик прерывания, он должен проделать обратную операцию — размаскировать линию сигнала запроса от соответствующего устройства. Например, для драйвера мыши PS/2-типа, обрабатывающего прерывание IRQ12 (ведомый контроллер, линия № 4), участок кода, в котором выполняется размаскирование, выглядит следующим образом:

```
; Прочитать регистр маски ведомого контроллера
        in      AL,0A1h
; Обнулить четвертый разряд маски
        and     AL,11101111b
; Вставить задержку
        jmp short $+2
        jmp short $+2
; Записать маску обратно в регистр
        out     0A1h,AL
```

Обработка сигнала прерывания выполняется контроллером следующим образом. Вначале выполняется проверка маски, и если сигнал не запрещен, то происходит проверка его приоритета. Если нет других сигналов или данный запрос имеет наивысший приоритет, то контроллер посылает микропроцессору сигнал запроса INT. Если маскируемые прерывания разрешены (установлен флаг IF в регистре флагов процессора), то процессор выдает контроллеру сигнал подтверждения прерывания INTA. После получения сигнала INTA контроллер выдает процессору номер вектора прерывания и переводит запрос в разряд обслуживаемых (соответствующий бит в регистре поступивших запросов контроллера сбрасывается, а бит в регистре обслуживаемых запросов устанавливается). Процессор начинает обработку прерывания с того, что записывает в стек содержимое регистра флагов и сбрасывает флаг IF.

В результате перечисленных выше действий в момент начала обработки прерывания все маскируемые прерывания в процессоре запрещены, а контроллер не пропускает запросы прерываний, приоритеты которых ниже, чем у запроса, обрабатываемого процессором. Кроме того, следующие прерывания от устройства, запрос которого поступил на обработку, также будут заблокированы контроллером. Процессор должен разрешить обработку маскируемых прерываний (установить флаг IF командой STI), как только будет выполнен критический участок кода процедуры обработки прерывания, то есть группа операций, которую прерывать нельзя. Часто процедура вообще не содержит критических участков, и тогда команду STI следует установить в самом начале обработчика прерывания. С другой стороны, весь код процедуры может не допускать прерывания: в этом случае команда STI не используется, а флаг прерываний автоматически восстанавливается при выходе из процедуры по команде IRET.

Разрешить контроллеру обрабатывать прерывания с таким же или более низким приоритетом следует как можно раньше, чтобы не возникала угроза потери информации от низкоприоритетных устройств. Как только будет завершен участок кода, не допускающий повторного прерывания от того же устройства, необходимо послать контроллеру команду EOI (End Of Interrupt). Регистр команд ведущей микросхемы доступен для записи через порт 20h, регистр команд ведомой — через порт A0h. Например, в процедуру обработчика прерывания от клавиатуры обязательно должна быть включена последовательность команд:

```
; Записать в регистр AL код команды EOI
        mov     AL,20h
```

```
; Послать команду EOI в ведущую микросхему
        out     20h,AL
```

Прерывание, поступившее через ведомую микросхему, блокирует также и обработку всех прерываний с более низкими приоритетами в ведущей. Поэтому процедура обработки такого прерывания должна посылать команду EOI не только в ведомую, но и в ведущую микросхему:

```
; Записать в регистр AL код команды EOI
        mov     AL,20h
; Послать команду EOI в ведомую микросхему
        out     0A0h,AL
; Послать команду EOI в ведущую микросхему
        out     20h,AL
```

Непосредственная работа с контроллером клавиатуры

Реальная необходимость в непосредственной работе с клавиатурой возникает в том случае, если вы создаете программу, которая переводит процессор из реального режима в защищенный, а затем выполняет в защищенном режиме всю дальнейшую работу. Переход в защищенный режим приводит к тому, что функции BIOS, рассчитанные на реальный режим, становятся непригодными для использования.

Для управления работой клавиатуры в машинах типа IBM AT и PS/2 использовался микроконтроллер Intel 8042. Кроме клавиатуры, этот контроллер управлял также координатным устройством, в качестве которого обычно использовалась мышь типа PS/2. С целью обеспечения совместимости нового аппаратного обеспечения со старыми программами все современные наборы микросхем, предназначенные для изготовления системных плат (так называемые чипсеты), вынужденно повторяют в своей структуре особенности контроллера i8042 [57, 69, 70, 79, 98].

Писать программы для клавиатурного контроллера i8042 и встроенного микропроцессора клавиатуры нет необходимости и возможности, да и бесполезно, поскольку соответствующие программы уже «намертво» записаны в ПЗУ контроллеров. Поэтому клавиатурный контроллер на системной плате и микропроцессор клавиатуры могут выполнять только те операции, которые заложены в них разработчиками аппаратуры и внесены в соответствующие наборы команд. Самопроверка контроллера и программирование основных

параметров его работы, а также самотестирование и программирование параметров клавиатуры (и мыши PS/2) производятся при включении компьютера и могут быть изменены с помощью прерываний BIOS, пока процессор еще не переведен из реального режима в защищенный. В защищенном режиме необходимо выполнять только два типа операций: считывание кодов нажимаемых клавиш и зажигание/гашение светодиодов на клавиатуре. Этим операциям уделяется основное внимание, а бесполезная на практике часть информации о программировании контроллера будет опущена (при необходимости дополнительные сведения можно найти в литературе [6, 7, 14, 30]). В данной главе мы рассмотрим только функционирование клавиатурного контроллера и клавиатуры (управление мышью PS/2 будет рассматриваться в главе 5 «Работа с мышью»).

Управление работой контроллера и обмен данными с мышью и клавиатурой осуществляются при помощи трех регистров: регистра состояния, регистра команд и регистра данных. Кроме того, при поступлении информации от клавиатуры контроллер i8042 вырабатывает прерывание IRQ1, а при приеме данных от мыши — IRQ12. Интерфейсы клавиатуры и мыши аналогичны, наборы команд управления также имеют некоторое сходство. Упрощенная схема подключения к компьютеру клавиатуры и мыши типа PS/2 показана на рис. 1.5.

Регистр состояния доступен только для считывания через порт 64h. Формат регистра показан на рис. 1.6. Значение разрядов регистра следующее:

- бит 0 — признак наличия данных в выходном буфере (0 — буфер пуст, 1 — буфер заполнен);
- бит 1 — признак наличия данных во входном буфере (0 — буфер пуст, 1 — буфер заполнен);
- бит 2 — признак типа последнего общесистемного сброса (0 — сброс по включении питания, 1 — программный сброс);
- бит 3 — признак записи команды (0 — последняя операция записи являлась операцией записи данных, 1 — записи команды);
- бит 4 — состояние «замка» клавиатуры (0 — клавиатура блокирована, 1 — не блокирована);
- бит 5 — признак ошибки тайм-аута передачи в AT-режиме (0 — нормальное завершение передачи, 1 — произошла ошибка); признак наличия данных в выходном буфере мыши в PS/2-режиме (0 — буфер пуст, 1 — буфер заполнен);

- бит 6 — признак ошибки тайм-аута приема в AT-режиме (0 — нормальное завершение приема, 1 — произошла ошибка); общий признак ошибки тайм-аута при приеме или передаче данных в PS/2-режиме (0 — нормальное завершение операции, 1 — произошла ошибка);
- бит 7 — признак возникновения ошибки паритета при приеме или передаче данных (0 — нет ошибки, 1 — обнаружена ошибка по четности).

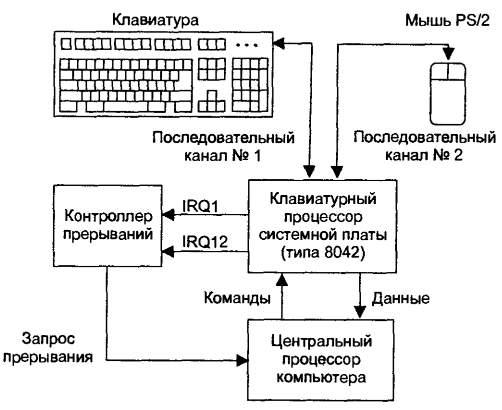


Рис. 1.5. Упрощенная схема подключения клавиатуры и мыши типа PS/2 к компьютеру

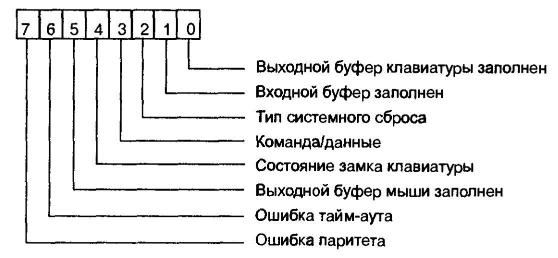


Рис. 1.6. Формат регистра состояния контроллера клавиатуры

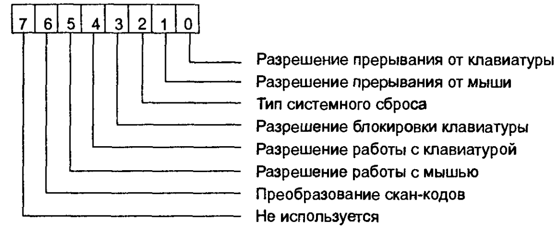


Рис. 1.7. Формат регистра команд контроллера клавиатуры

Регистр команд доступен для записи через порт 64h. Формат регистра показан на рис. 1.7. Разряды регистра имеют следующее назначение:

- бит 0 — управление выдачей сигнала прерывания по готовности данных в выходном буфере клавиатуры (0 — генерация прерывания запрещена, 1 — разрешена);
- бит 1 — управление выдачей сигнала прерывания по готовности данных в выходном буфере мыши (0 — генерация прерывания запрещена, 1 — разрешена);
- бит 2 — установка признака системного сброса (значение, записанное в этот разряд, переносится в разряд 2 регистра состояния);
- бит 3 — управление функцией блокировки клавиатуры (0 — функция блокировки разрешена, 1 — функция запрещена и блокировка игнорируется);
- бит 4 — управление интерфейсом клавиатуры (0 — обмен данными с клавиатурой разрешен, 1 — запрещен);
- бит 5 — признак типа протокола передачи данных в AT-режиме (0 — протокол IBM); управление интерфейсом мыши в PS/2-режиме (0 — обмен данными с мышью разрешен, 1 — запрещен);
- бит 6 — преобразование скан-кодов в PC-совместимые (0 — выключено, 1 — включено); по умолчанию имеет значение 1, то есть скан-коды преобразуются в PC-совместимые;
- бит 7 — не используется (зарезервирован).

Коды команды управления приведены в табл. 1.8. Некоторые из команд двухбайтные, то есть имеют данные: первый байт (код команды) в этом случае записывается в регистр команд, а следующий — в регистр данных. Двухбайтными являются команды 60h, 01h–04h.

Клавиатурный контроллер системной платы может работать в двух различных режимах: в АТ-режиме он может управлять только клавиатурой, в PS/2-режиме – клавиатурой и мышью. Регистр команд обеспечивает грубую установку режима работы клавиатуры и мыши. Кроме того, режимами работы также можно управлять, посылая в регистр определенные кодовые комбинации.

Регистр данных доступен для записи и считывания через порт 60h. В режиме считывания он служит для приема информации от клавиатуры и мыши. В режиме записи регистр данных служит для передачи команд клавиатуре, координатному устройству (мыши PS/2-типа) и клавиатурному контроллеру системной платы i8042. Какого-либо особого формата данный регистр не имеет.

Таблица 1.8. Команды управления контроллером клавиатуры и мыши

Код	Назначение команды
20h	Чтение содержимого регистра команд
60h	Запись командного байта в регистр команд
A1h	Чтение номера версии контроллера
A4h	Проверка «пароля» (признака подключения клавиатуры). При успешном выполнении команды выдается строка «F1»
A7h	Запретить работу с мышью (команда выполняется только в режиме PS/2, в АТ-режиме игнорируется)
A8h	Разрешить работу с мышью (команда выполняется только в режиме PS/2, в АТ-режиме игнорируется)
A9h	Тест интерфейса мыши (при успешном выполнении команды генерируется значение 00h)
AAh	Самотестирование контроллера (при успешном выполнении команды генерируется значение 55h)
ABh	Тест интерфейса клавиатуры (при успешном выполнении команды генерируется значение 00h)
ADh	Запретить работу с клавиатурой
AEh	Разрешить работу с клавиатурой
AFh	Выдать номер версии
C0h	Чтение байта состояния входного порта (порта № 1)
D0h	Чтение байта состояния выходного порта (порта № 2)
D1h	Запись байта в выходной порт (порт № 2)
D2h	Запись байта в выходной буфер клавиатуры
D3h	Запись байта в выходной буфер мыши
D4h	Передать мыши байт данных

Клавиатура может работать в трех различных режимах, которые отличаются друг от друга наборами скан-кодов, выдаваемых клавиатурой. Набор № 1 предназначен для компьютеров типа IBM «PS/2 30», набор № 2 — для «PC/AT», «PS/2 50» и «PS/2 60», набор № 3 — для «PS/2 80». Для переключения режимов работы используется команда F0h. Однако программисты никогда не сталкиваются непосредственно с наборами кодов, выдаваемыми клавиатурой, а имеют дело с преобразованным набором, выдаваемым клавиатурным процессором системной платы (так что переключение режима — обычно не только бесполезная, но даже вредная операция). Если потребуется использование типовой компьютерной клавиатуры в каком-либо самодельном устройстве, то стандартные наборы скан-кодов можно посмотреть в специальной литературе, например в [6] или [57].

При включении питания клавиатура устанавливается в режим № 2, причем для всех клавиш разрешена посылка кодов нажатия и отпускания, а также разрешен автоповтор. Самотестирование после включения выполняется в следующем порядке:

- включаются все индикаторы;
- тестируется встроенный микроконтроллер клавиатуры;
- тестируется оперативная память клавиатуры;
- все индикаторы выключаются;
- клавиатура выдает компьютеру результат самотестирования.

Когда разрешен опрос клавиш, клавиатура передает компьютеру скан-коды, сообщающие об изменении их состояния (под изменением состояния подразумевается нажатие, длительное удержание или отпускание). Кроме того, возможна посылка следующих специализированных кодов в ответ на команду или при возникновении неисправностей:

- 00h — возникла ошибка переполнения (в режиме № 2 или № 3);
- AAh — самотестирование прошло успешно;
- последовательность ABh, B3h — идентификатор клавиатуры;
- EEh — ответ на эхо-команду EEh;
- FAh — подтверждение приема информации (команда ACK);
- FCh — в процессе самотестирования обнаружена неисправность;
- FEh — запрос на повторную передачу команды со стороны компьютера, выдается в случае возникновения ошибки передачи;

- FFh — возникла ошибка переполнения (в режиме № 1).

Список команд управления встроенным процессором клавиатуры приведен в табл. 1.9.

Таблица 1.9. Команды управления встроенным процессором клавиатуры

Код	Назначение команды
EDh	Установить состояние индикаторов (светодиодов)
EEh	Эхо-диагностика
F0h	Изменить режим работы (набор скан-кодов)
F2h	Выдать идентификационную последовательность кодов
F3h	Установить параметры режима автоповтора
F4h	Разрешить передачу данных
F5h	Установить значение параметров работы, используемое по умолчанию, и запретить опрос клавиш
F6h	Установить значение параметров работы, используемое по умолчанию
F7h	Разрешить режим автоповтора для всех клавиш
F8h	Разрешить для всех клавиш посылку кодов нажатия и отпускания
F9h	Разрешить для всех клавиш посылку только кодов нажатия
FAh	Разрешить для всех клавиш режим автоповтора, посылку кодов нажатия и отпускания
FBh	Разрешить режим автоповтора заданной клавиши
FCh	Разрешить для заданной клавиши посылку только кода нажатия
FDh	Разрешить для заданной клавиши посылку кодов нажатия и отпускания
FEh	Повторить передачу пакета данных
FFh	Произвести сброс

Рассмотрим приведенные в таблице команды более подробно.

Команда EDh предназначена для включения и выключения индикаторов состояния клавиатуры (светодиодов Num Lock, Caps Lock и Scroll Lock). Команда двухбайтная — байт данных, следующий за кодом команды, содержит информацию, управляющую включением и выключением светодиодов. Формат байта данных следующий:

- бит 0 — состояние индикатора Scroll Lock (0 — выключен, 1 — включен);
- бит 1 — состояние индикатора Num Lock (0 — выключен, 1 — включен);

- бит 2 — состояние индикатора Caps Lock (0 — выключен, 1 — включен);
- биты 3–7 не используются.

На команду управления индикаторами клавиатура реагирует следующим образом:

- посылает байт ACK в компьютер;
- останавливает опрос клавиш;
- принимает от компьютера байт данных, определяющий новое состояние индикаторов;
- посылает байт ACK в компьютер;
- устанавливает индикаторы в заданное состояние.

Команда EEh предназначена для эхо-диагностики (в ответ микропроцессор клавиатуры также выдает код EEh). Обычно команда используется при возникновении ошибок в работе клавиатуры. Посылка команды EEh после выполнения сброса клавиатуры позволяет определить, восстановился ли нормальный режим функционирования. Неправильный ответ на эхо-команду означает, что процессор клавиатуры «зациклился» и привести систему в нормальное состояние можно только отключением питания.

Реакция клавиатуры на команду EEh следующая:

- компьютеру выдается код EEh;
- продолжается работа в том режиме, который был установлен до поступления команды.

Команда F0h позволяет установить один из трех наборов скан-кодов, используемых для передачи данных от клавиатуры к клавиатурному процессору материнской платы. Команда двухбайтная — байт данных должен следовать за байтом команды и содержать номер устанавливаемого набора (1, 2 или 3). Клавиатура реагирует на команду следующим образом:

- посылает байт ACK в компьютер;
- очищает выходной буфер;
- принимает байт данных, содержащий номер устанавливаемого режима;
- устанавливает новый режим.

Команда F2h служит для идентификации типа устройства, подключенного к клавиатурному разъему компьютера, то есть для провер-

ки того, что в разъем включена именно клавиатура, а не присоединено по ошибке какое-либо другое устройство вроде мыши или джойстика. Реакция клавиатуры на команду идентификации следующая:

- клавиатура посылает компьютеру байт ACK;
- приостанавливается опрос клавиш;
- клавиатура посылает компьютеру код ABh, затем — код 83h;
- опрос клавиш возобновляется.

Команда F3h позволяет установить значение частоты и задержки автоповтора. Команда является двухбайтной — байт данных следует за кодом команды и имеет следующую структуру:

- биты 0–4 — код частоты автоповтора (см. табл. 1.5);
- биты 5–6 — код задержки автоповтора (см. табл. 1.4);
- бит 7 — не используется (всегда должен иметь значение 0).

Клавиатура реагирует на команду установки параметров автоповтора следующим образом:

- посылает байт ACK в компьютер;
- останавливает опрос клавиш;
- принимает от компьютера байт данных, определяющий новое значение параметров автоповтора;
- посылает байт ACK в компьютер;
- устанавливает заданные параметры автоповтора.

Команда F4h разрешает микропроцессору клавиатуры возобновить опрос клавиш и передачу данных компьютеру. Реакция клавиатуры на команду F4h следующая:

- клавиатура посылает байт ACK в компьютер;
- производится очистка выходного буфера;
- если какая-либо клавиша работала в режиме автоповтора, то автоповтор прекращается;
- возобновляется опрос клавиш.

Команда F5h устанавливает значения параметров клавиатуры, принятые по умолчанию, и останавливает сканирование (опрос клавиш и передачу данных об изменении их состояния компьютеру). Микропроцессор клавиатуры реагирует на команду F5h следующим образом:

- посылает байт ACK в компьютер;
- стирает выходной буфер;

- разрешает для всех клавиш (кроме Pause) автоповтор и посылку кодов нажатия и отпускания;
- устанавливает используемое по умолчанию значение параметров автоповтора;
- прекращает автоповтор, если какая-то клавиша работала в режиме автоповтора;
- прекращает опрос клавиш;
- переходит в режим ожидания дальнейших инструкций.

Команда F6h устанавливает значения параметров клавиатуры, принятые по умолчанию. Клавиатура реагирует следующим образом:

- посылает байт ACK в компьютер;
- стирает выходной буфер;
- разрешает для всех клавиш (кроме Pause) автоповтор и посылку кодов нажатия и отпускания;
- устанавливает используемое по умолчанию значение параметров автоповтора.

Команды F7h–FAh определяют тип реакции клавиатурного процессора на нажатие, удержание и отпускание клавиш. Команды воздействуют сразу на все клавиши клавиатуры: команда F7h включает режим автоповтора, команда F8h разрешает посылку кодов нажатия и отпускания, команда F9h — только кодов нажатия, а команда FAh включает автоповтор и разрешает посылку кодов нажатия и отпускания. Реакция клавиатуры на любую из этих команд следующая:

- клавиатура посылает байт ACK в компьютер;
- очищает выходной буфер;
- устанавливает заданный тип реакции на нажатие, удержание и отпускание клавиш.

Команды FBh–FDh определяют тип реакции клавиатурного процессора на нажатие, удержание и отпускание конкретной, указанной в команде клавиши. Эти команды двухбайтовые — второй байт задает идентификатор клавиши, для которой производится установка индивидуального режима. Команда FBh включает режим автоповтора, команда FCh разрешает посылку кодов нажатия и отпускания, команда FDh — только кодов нажатия. Данные команды выполняются только в том случае, если клавиатура функционирует в режиме № 3. Клавиатура реагирует на любую из них следующим образом:

- посылает байт ACK в компьютер;
- очищает выходной буфер;

- ожидает поступления кода-идентификатора клавиши;
- устанавливает для заданной клавиши заданный тип реакции на нажатие, удержание и отпускание.

Режим работы клавиатуры устанавливается при запуске компьютера процедурами BIOS, и после этого изменять его обычно нет необходимости. Единственная команда, которую драйвер периодически посылает клавиатуре, — команда переключения светодиодов EDh. Данная команда является ответной реакцией драйвера на нажатие клавиш Num Lock, Caps Lock и Scroll Lock.

Как уже отмечалось выше, набор скан-кодов, которые выдает клавиатурный процессор системной платы, отличается и от набора скан-кодов, которые используют процедуры BIOS. На рис. 1.8 показаны коды, присвоенные клавишам основной и функциональной групп, на рис. 1.9 — коды дополнительной клавиатуры, а на рис. 1.10 — коды цифровой клавиатуры.

01		3B	3C	3D	3E	3F	40	41	42	43	44	45	46

29	02	03	04	05	06	07	08	09	0A	0B	0C	0D	2B	0E
0F	10	11	12	13	14	15	16	17	18	19	1A	1B		
3A	1E	1F	20	21	22	23	24	25	26	27	28	1C		
2A	2C	2D	2E	2F	30	31	32	33	34	35	36			
1D		38	39	E0,38		E0,1D								

Рис. 1.8. Скан-коды клавиш основной и функциональной клавиатур

E0,2A, E0,37	46	E1,1D,45, E1,9D,C5

E0,2A, E0,52	E0,2A, E0,47	E0,2A, E0,49
E0,2A, E0,53	E0,2A, E0,4F	E0,2A, E0,51

	E0,2A, E0,48	
E0,2A, E0,4B	E0,2A, E0,50	E0,2A, E0,4D

Рис. 1.9. Скан-коды клавиш дополнительной клавиатуры

<table>
<tr><td>45</td><td>E0,35</td><td>37</td><td>4A</td></tr>
<tr><td>47</td><td>48</td><td>49</td><td rowspan="2">4E</td></tr>
<tr><td>4B</td><td>4C</td><td>4D</td></tr>
<tr><td>4F</td><td>50</td><td>51</td><td rowspan="2">E0,1C</td></tr>
<tr><td colspan="2">52</td><td>53</td></tr>
</table>

Рис. 1.10. Скан-коды клавиш цифровой клавиатуры

Нажатие клавиши приводит к передаче от клавиатуры к компьютеру одного символа или последовательности символов (от двух до шести). При нажатии обычных клавиш (алфавитно-цифровых или функциональных) передается только один байт, содержащий скан-код. Последовательности генерируются для клавиш, которые отсутствовали в 84-кнопочной клавиатуре XT-типа, и состоят из кодовых пар, причем каждая пара начинается с кода E0h, а во втором байте передается скан-код. Последовательность из четырех байт (двух пар) передается в том случае, если нажата дополнительная клавиша, заменяющая собой нажатие определенной последовательности обычных клавиш. Специфическая последовательность из шести байт генерируется только в одном случае — при нажатии клавиши Pause.

При отпускании клавиши клавиатура также посылает в компьютер скан-код, но старший (знаковый) разряд кода при этом устанавливается в единицу. Отпускание клавиши, выдающей пару кодов, можно отличить от нажатия по второму символу пары (первым кодом в паре по прежнему является E0h, а у скан-кода при отпускании будет установлен старший разряд). При отпускании дополнительных клавиш генерируются две пары кодов, но порядок этих пар является обратным тому, который генерируется при их нажатии, и установлены старшие разряды скан-кодов. При отпускании клавиши Pause клавиатура никакой информации в компьютер не передает.

Например, при нажатии клавиши **Пробел** вырабатывается код 39h, а при отпускании — код B9h. При нажатии клавиши 1 на основной клавиатуре вырабатывается код 02h, а при отпускании — код 82h; нажатие клавиши 1 на цифровой клавиатуре порождает код 4Fh, а отпускание — код CFh. При нажатии правой клавиши Ctrl вырабатывается последовательность E0, 1Dh, а при отпускании — последо-

вательность E0h, 9Dh. Нажатие Insert порождает последовательность кодов E0h, 2Ah, E0h, 52h, а отпускание — последовательность E0h, D2h, E0h, AAh. Нажатие клавиши Pause приводит к выдаче последовательности E1h, 1Dh, 45h, E1h, 9Dh, C5h, а при отпускании данной клавиши никаких кодов не вырабатывается вообще.

Для поддержки расширенного интерфейса управления конфигурацией и питанием (Advanced Configuration and Power Interface, сокращенно ACPI) на клавиатуру были добавлены три клавши [71]:

- Power (Выключить питание) — вырабатывает последовательность E0h, 5Eh;
- Sleep (Переключить систему в спящий режим) — вырабатывает последовательность E0h, 5Fh;
- Wake (Разбудить систему) — вырабатывает последовательность E0h, 63h.

Появились в продаже также так называемые «Мультимедийные клавиатуры» с целой группой дополнительных клавиш:

- Next Track — вырабатывает последовательность E0h, 19h;
- Previous Track — вырабатывает последовательность E0h, 10h;
- Stop — вырабатывает последовательность E0h, 24h;
- Play/Pause — вырабатывает последовательность E0h, 22h;
- Mute — вырабатывает последовательность E0h, 20h;
- Volume Up — вырабатывает последовательность E0h, 30h;
- Volume Down — вырабатывает последовательность E0h, 2Eh;
- Media Select — вырабатывает последовательность E0h, 6Dh;
- E-Mail — вырабатывает последовательность E0h, 6Ch;
- Calculator — вырабатывает последовательность E0h, 21h;
- My Computer — вырабатывает последовательность E0h, 6Bh;
- WWW Search — вырабатывает последовательность E0h, 65h;
- WWW Home — вырабатывает последовательность E0h, 32h;
- WWW Back — вырабатывает последовательность E0h, 6Ah;
- WWW Forward — вырабатывает последовательность E0h, 69h;
- WWW Stop — вырабатывает последовательность E0h, 68h;
- WWW Refresh — вырабатывает последовательность E0h, 67h;
- WWW Favorites — вырабатывает последовательность E0h, 66h.

ПРИМЕЧАНИЕ

Местоположение дополнительных клавиш на клавиатуре не стандартизировано. Обычно мультимедийные клавиши размещаются в один ряд над функциональными клавишами, а клавиши ACPI находятся между клавишами дополнительной клавиатуры.

Примеры программ, работающих непосредственно с контроллером клавиатуры, приведены в листингах 1.6 и 1.7. Программа KeyboardTest из листинга 1.6 предназначена для отображения на экран потока кодов, поступающих с клавиатуры. Она позволяет наблюдать реакцию клавиатуры на нажатие, удержание и отпускание клавиш различных типов. Процедура обработки клавиатурного прерывания IRQ1 KeyboardInterrupt отображает принятый от клавиатуры код на экран в шестнадцатеричном представлении, а затем производит подготовку к выводу следующего кода (изменяет значение текущих экранных координат ScreenColumn и ScreenString). Для установки вектора прерывания на эту процедуру служит подпрограмма SetKeyboardInterrupt, а для восстановления старого вектора прерывания после завершения работы основной программы используется подпрограмма RestoreOldKeyboardInterrupt.

Листинг 1.6. Отображение потока данных от клавиатуры в шестнадцатеричных кодах

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Счетчик операций нажатия/отпускания клавиш
PressCounter DW ?
; Область сохранения старого вектора прерывания клавиатуры
OldKeyboardInterruptOffset  DW ?
OldKeyboardInterruptSegment DW ?
; Текстовые сообщения
```

продолжение ⇨

Листинг 1.6 *(продолжение)*

```
Text DB LIGHTGREEN,0,19
     DB "Непосредственный прием данных с клавиатуры",0
     DB YELLOW,4,0,"Отображение потока данных "
     DB "в шестнадцатеричных кодах:",0
     DB LIGHTBLUE,24,29,"Нажмите любую клавишу",0
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC KeyboardTest
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 3,Text
; Установить белый цвет символов и черный фон
        mov     [TextColorAndBackground],WHITE
; Установить начальную позицию для вывода кодов
; в нулевой колонке пятой строки
        mov     [ScreenString],5
        mov     [ScreenColumn],0
; Инициализировать счетчик операций
; нажатия/отпускания клавиш
        mov     [PressCounter],0
; Установить новый обработчик прерывания
        call    SetKeyboardInterrupt

; Ожидать, пока не будет произведено 200
; операций нажатия/отпускания
@@Next: cmp     [PressCounter],200
        jb      @@Next

; Восстановить исходный обработчик прерывания
        call    RestoreOldKeyboardInterrupt
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Передать управление DOS
        mov     AH,4Ch
        int     21h
```

```
ENDP KeyboardTest

;*******************************************
;* НОВЫЙ ОБРАБОТЧИК ПРЕРЫВАНИЯ ОТ КЛАВИАТУРЫ *
;*******************************************
proc KeyboardInterrupt far
        pusha
        push    DS
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
; Получить скан-код
        in      AL,60h
; Разрешить прерывания с более низким приоритетом
        push    AX
        mov     AL,20h  ;команда EOI
        out     20h,AL
        pop     AX
        sti             ;разрешить прерывания
; Отобразить на экране монитора принятый от клавиатуры
; код в шестнадцатеричном представлении
        ; Отобразить принятый байт
        call    ShowHexByte
        ; Перевести текущую позицию на 2 символа влево
        ; (подготовка для вывода следующего кода)
        add     [ScreenColumn],2
        ; Проверить пересечение правой границы экрана
        cmp     [ScreenColumn],80
        jb      @@EndShowCode
        ; Если достигнута правая граница
        ; экрана - перейти на следующую строку
        sub     [ScreenColumn],80
        inc     [ScreenString]
@@EndShowCode:
; Завершение обработки прерывания
        ; Увеличить значение счетчика нажатий/отпусканий
        inc     [PressCounter]
        pop     DS
        popa
        iret
endp KeyboardInterrupt

;**************************************************
;* УСТАНОВИТЬ НОВЫЙ ОБРАБОТЧИК ПРЕРЫВАНИЯ КЛАВИАТУРЫ *
;**************************************************
PROC SetKeyboardInterrupt NEAR
        pusha
        push    ES
        mov     AX,0
        mov     ES,AX
```

продолжение ⇩

Листинг 1.6 (*продолжение*)

```
; Запомнить прежний вектор обработчика
; прерывания клавиатуры
        mov     AX,[ES:9*4]
        mov     [OldKeyboardInterruptOffset],AX
        mov     AX,[ES:9*4+2]
        mov     [OldKeyboardInterruptSegment],AX
; Установка вектора прерывания на обработчик клавиатуры
        cli                     ;запретить прерывания
        mov     AX,offset KeyboardInterrupt
        mov     [ES:9*4],AX
        mov     AX,CS
        mov     [ES:9*4+2],AX
        sti                     ;разрешить прерывания
        pop     ES
        popa
        ret
ENDP SetKeyboardInterrupt

;*****************************************
;* ВОССТАНОВИТЬ ИСХОДНЫЙ ВЕКТОР ПРЕРЫВАНИЯ *
;*****************************************
PROC RestoreOldKeyboardInterrupt NEAR
        pusha
; Настроить регистр ES на таблицу векторов прерываний
        push    ES
        mov     AX,0
        mov     ES,AX
; Восстановить прежний вектор обработчика прерывания
        cli
        mov     AX,[OldKeyboardInterruptOffset]
        mov     [ES:9*4],AX
        mov     AX,[OldKeyboardInterruptSegment]
        mov     [ES:9*4+2],AX
        sti
        pop     ES
        popa
        ret
ENDP RestoreOldKeyboardInterrupt
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"

END
```

Программа KeyboardDriver из листинга 1.7 демонстрирует выполнение следующих типовых операций над данными, поступающими от клавиатурного контроллера системной платы:

- преобразование скан-кодов в ASCII-коды;
- обработка управляющих клавиш;
- управление светодиодами клавиатуры.

Кроме процедур общего назначения, в программе KeyboardDriver используются следующие подпрограммы:

- процедура KeyboardInterrupt принимает данные от клавиатуры, обрабатывает их и при необходимости запоминает введенный символ в выделенной для него ячейке или переключает состояние светодиодов на клавиатуре;
- процедура SetKeyboardInterrupt служит для установки вектора прерывания от клавиатуры на подпрограмму KeyboardInterrupt;
- процедура RestoreOldKeyboardInterrupt служит для восстановления прежнего вектора прерывания после завершения работы программы;
- процедура Wait8042BufferEmpty выполняет цикл ожидания освобождения буфера контроллера клавиатуры.

Листинг 1.7. Упрощенный вариант драйвера клавиатуры

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Счетчик операций нажатия/отпускания клавиш
PressCounter DW ?
; Область сохранения старого вектора
; прерывания клавиатуры
OldKeyboardInterruptOffset  DW ?
OldKeyboardInterruptSegment DW ?
; Байт состояния клавиатуры (бит 0 - ScrollLock,
; бит 1 - NumLock, бит 2 - CapsLock, бит 3 - левый
```

продолжение ⇗

Листинг 1.7 (*продолжение*)

```
; Shift, бит 4 - правый Shift)
KeyboardStatus DB 0
; Байт состояния светодиодов (бит 0 - ScrollLock,
; бит 1 - NumLock, бит 2 - CapsLock)
LEDStatus DB 0
; Признак ввода очередного символа
CharInputFlag DB 0
; ASCII-код введенного символа
ASCIICode DB 0
; Текстовые сообщения
Text DB LIGHTGREEN,0,21
     DB "Упрощенный вариант драйвера клавиатуры",0
     DB YELLOW,10,0
     DB "Отображение потока данных в ASCII-кодах:",0
     DB LIGHTGREY,24,29,"Нажмите любую клавишу",0
; Перекодировочные таблицы (для алфавитно-цифровой
; клавиатуры)
label RusNorm byte
  DB  0 , 27,'1','2','3','4','5','6'
  DB '7','8','9','0','-','=', 8 , 9
  DB 'й','ц','у','к','е','н','г'
  DB 'ш','щ','з','х','ъ', 13, 0
  DB 'ф','ы','в','а','п','р','о'
  DB 'л','д','ж','э','ё', 0 ,'\'
  DB 'я','ч','с','м','и','т','ь'
  DB 'б','ю','.', 0 ,'*', 0 ,' '
label RusNormShift byte
  DB  0 , 27,'!','"','№',';','%',':'
  DB '?','*','(',')','_','+', 8 , 9
  DB 'Й','Ц','У','К','Е','Н','Г'
  DB 'Ш','Щ','З','Х','Ъ', 13, 0
  DB 'Ф','Ы','В','А','П','Р','О'
  DB 'Л','Д','Ж','Э','Ё', 0 ,'/'
  DB 'Я','Ч','С','М','И','Т','Ь'
  DB 'Б','Ю',',', 0 ,'*', 0 ,' '
label RusCaps byte
  DB  0 , 27,'1','2','3','4','5','6'
  DB '7','8','9','0','-','=', 8 , 9
  DB 'Й','Ц','У','К','Е','Н','Г'
  DB 'Ш','Щ','З','Х','Ъ', 13, 0
  DB 'Ф','Ы','В','А','П','Р','О'
  DB 'Л','Д','Ж','Э','Ё', 0 ,'\'
  DB 'Я','Ч','С','М','И','Т','Ь'
  DB 'Б','Ю','.', 0 ,'*', 0 ,' '
label RusCapsShift byte
  DB  0 , 27,'!','"','№',';','%',':'
  DB '?','*','(',')','_','+', B , 9
  DB 'й','ц','у','к','е','н','г'
  DB 'ш','щ','з','х','ъ', 13, 0
```

```
  DB 'ф','ы','в','а','п','р','о'
  DB 'л','д','ж','э','ё', 0 ,'/'
  DB 'я','ч','с','м','и','т','ь'
  DB 'б','ю','.', 0 ,'*', 0 ,' '
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC KeyboardDriver
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Вывести текстовые сообщения на экран
        MShowColorText 3,Text
; Инициализировать счетчик операций
; нажатия/отпускания клавиш
        mov     [PressCounter],0
; Установить новый обработчик прерывания
        call    SetKeyboardInterrupt
; Установить курсор в начало строки
        mov     [ScreenString],12
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Настроить ES:DI на видеопамять
        mov     AX,0B800h ;текстовый видеосегмент
        mov     ES,AX
        mov     AH,WHITE  ;белый цвет, черный фон
        ; Выполнять вывод символов в двенадцатой строке
        mov     DI,12*160
; Ввести и отобразить на экране 80 символов
@@NextChar:
        cmp     [CharInputFlag],0 ;Введен символ?
        je      @@NextChar
        ; Сбросить флаг ввода символа
        mov     [CharInputFlag],0
        ; Отобразить символ
        mov     AL,[ASCIICode]
        stosw
        ; Передвинуть курсор в следующую позицию
        inc     [ScreenColumn]
        call    SetCursorPosition
        ; Увеличить счетчик
        inc     [PressCounter]
        ; Проверить условие завершения цикла
```

продолжение ⇗

Листинг 1.7 (*продолжение*)

```
        cmp     [PressCounter],80
        jb      @@NextChar

; Восстановить исходный обработчик прерывания
        call    RestoreOldKeyboardInterrupt
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Передать управление DOS
        mov     AH,4Ch
        int     21h
ENDP KeyboardDriver

;*********************************************
;* НОВЫЙ ОБРАБОТЧИК ПРЕРЫВАНИЯ ОТ КЛАВИАТУРЫ *
;*********************************************
proc KeyboardInterrupt far
;Сохранить регистры
        pusha
        push    DS
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
; Получить скан-код
        in      AL,60h
        ;Разрешить прерывания
        push    AX          ;сохранить скан-код
        mov     AL,20h      ;послать команду EDI
        out     20h,AL      ;в ведущий контроллер
        pop     AX          ;восстановить скан-код
        sti

; Игнорировать код подтверждения команды АСК
        cmp     AL,0FAh     ;код подтверждения команды?
        je      @@End
; Обработать Shift
        cmp     AL,2Ah      ;левый Shift нажат
        je      @@LeftShiftON
        cmp     AL,0AAh     ;левый Shift отпущен
        je      @@LeftShiftOFF
        cmp     AL,36h      ;правый Shift нажат
        je      @@RightShiftON
        cmp     AL,0B6h     ;правый Shift отпущен
        je      @@RightShiftOFF
; Проверить нажатие клавиш статуса
        cmp     AL,3Ah      ;нажатие клавиши CapsLock?
        je      @@CapsLockLED_ON
        cmp     AL,45h      ;нажатие клавими NumLock?
        je      @@NumLockLED_DN
```

```
        cmp     AL,46h    ;нажатие клавиши ScrollLock?
        je      @@ScrollLockLED_ON

; Обработать алфавитно-цифровые клавиши
        cmp     AL,39h
        ja      @@End
        ; Использовать скан-код как индекс
        ; элемента в массиве перекодировки
        xor     BX,BX
        mov     BL,AL
        ; Выбрать массив перекодировки
        test    [KeyboardStatus],100b ;CapsLock активен?
        jnz     @@CapsON
        test    [KeyboardStatus],11000b ;Shift нажат?
        jnz     @@ShiftON1
        add     BX,offset RusNorm
        jmp short @@SaveChar
@@ShiftON1:
        add     BX,offset RusNormShift
        jmp short @@SaveChar
@@CapsON:
        test    [KeyboardStatus],11000b ;Shift нажат?
        jnz     @@ShiftON2
        add     BX,offset RusCaps
        jmp short @@SaveChar
@@ShiftON2:
        add     BX,offset RusCapsShift
@@SaveChar:
; Сохранить ASCII-код символа и установить флаг приема
        mov     AL,[BX]
        cmp     AL,32   ;это алфавитно-цифровой символ?
        jb      @@End
        mov     [ASCIICode],AL
        mov     [CharInputFlag],1
        jmp short @@End

; Установить признак Shift
@@LeftShiftON:
        or      [KeyboardStatus],1000b
        jmp short @@End
@@LeftShiftOFF:
        and     [KeyboardStatus],11110111b
        jmp short @@End
@@RightShiftON:
        or      [KeyboardStatus],10000b
        jmp short @@End
@@RightShiftOFF:
        and     [KeyboardStatus],11101111b
        jmp short @@End
```

продолжение ⇨

Листинг 1.7 (*продолжение*)

```
; При нажатии клавиши статуса зажечь или погасить
; соответствующий светодиод
@@CapsLockLED_ON:
        xor     [LEDStatus],100b
        jmp short @@SetKeyboardLED
@@NumLockLED_ON:
        xor     [LEDStatus],10b
        jmp short @@SetKeyboardLED
@@ScrollLockLED_ON:
        xor     [LEDStatus],1
@@SetKeyboardLED:
        ; Сбросить незначащие разряды
        and     [LEDStatus],111b
        ; Скопировать LEDStatus в KeyboardStatus
        and     [KeyboardStatus],11111000b
        mov     AL,[LEDStatus]
        or      [KeyboardStatus],AL
; Переустановить состояние светодиодов на клавиатуре
        call    Wait8042BufferEmpty
        ; Послать команду установки светодиодов
        mov     AL,0EDh
        out     60h,AL
        call    Wait8042BufferEmpty
        ; Передать байт состояния светодиодов
        mov     AL,[LEDStatus]
        out     60h,AL
@@End:  pop     DS
        popa
        iret
endp KeyboardInterrupt

;****************************************************
;* УСТАНОВИТЬ НОВЫЙ ОБРАБОТЧИК ПРЕРЫВАНИЯ КЛАВИАТУРЫ *
;****************************************************
PROC SetKeyboardInterrupt NEAR
        pusha
        push    ES
        mov     AX,0
        mov     ES,AX
; Запомнить прежний вектор обработчика
; прерывания клавиатуры
        mov     AX,[ES:9*4]
        mov     [OldKeyboardInterruptOffset],AX
        mov     AX,[ES:9*4+2]
        mov     [OldKeyboardInterruptSegment],AX
; Установка вектора прерывания на обработчик клавиатуры
        cli                 ;запретить прерывания
        mov     AX,offset KeyboardInterrupt
        mov     [ES:9*4],AX
        mov     AX,CS
```

```
        mov     [ES:9*4+2],AX
        sti                     ;разрешить прерывания
        pop     ES
        popa
        ret
ENDP SetKeyboardInterrupt

;*****************************************
;* ВОССТАНОВИТЬ ИСХОДНЫЙ ВЕКТОР ПРЕРЫВАНИЯ *
;*****************************************
PROC RestoreOldKeyboardInterrupt NEAR
        pusha
; Настроить регистр ES на таблицу векторов прерываний
        push    ES
        mov     AX,0
        mov     ES,AX
; Восстановить прежний вектор обработчика прерывания
        cli
        mov     AX,[OldKeyboardInterruptOffset]
        mov     [ES:9*4],AX
        mov     AX,[OldKeyboardInterruptSegment]
        mov     [ES:9*4+2],AX
        sti
        pop     ES
        popa
        ret
ENDP RestoreOldKeyboardInterrupt

;*********************************************
;*   ОЖИДАНИЕ ОЧИСТКИ ВХОДНОГО БУФЕРА I8042    *
;* При выходе из процедуры:                  *
;* флаг ZF установлен - нормальное завершение, *
;* флаг ZF сброшен - ошибка тайм-аута.        *
;*********************************************
proc Wait8042BufferEmpty near
        push    CX
        mov     CX,0FFFFh  ;задать число циклов
@@kb:   in      AL,64h     ;получить статус
        test    AL,10b     ;буфер i8042 свободен?
        loopnz  @@kb       ;если нет, то цикл
        pop     CX
        ; (если при выходе сброшен флаг ZF - ошибка)
        ret
endp Wait8042BufferEmpty
ENDS

; Подключить процедуры вывода данных на экран
include "listl_02.inc"

END
```

Следует иметь в виду, что приведенный пример был для наглядности сильно упрощен: драйвер обрабатывает только коды клавиш основной клавиатуры и управляющих клавиш Num Lock и Scroll Lock. Функциональные клавиши, клавиши дополнительной и цифровой клавиатур игнорируются. Из трех клавиш, состояние которых отображается светодиодами, влияние на коды вводимых символов оказывает только Caps Lock, а реакция на Num Lock и Scroll Lock сводится к зажиганию или гашению соответствующих индикаторов.

ВНИМАНИЕ

Запускать все приведенные в данном разделе примеры можно на любом AT-совместимом персональном компьютере с VGA-совместимым видеоконтроллером. Единственное ограничение: при помощи стандартного драйвера-русификатора DOS (или любого другого) перед запуском примеров должна быть загружена русская кодовая таблица, поскольку при выводе на экран используются сообщения на русском языке.

Глава 2 Недокументированные возможности процессоров Intel 80x86

У всех изготовителей микропроцессоров стало традицией предлагать описания регистров и команд через Интернет в виде pdf-файлов, но не давать при этом рекомендаций по их применению. Хорошо, если из названия (или описания) можно сделать совершенно определенные выводы о назначении команды или регистра. А если нет?

Столь же вредная традиция — не описывать в общедоступной документации режимы работы, которых современные процессоры имеют великое множество. Безусловным чемпионом в этой области является Intel — значительная часть потенциальных возможностей процессоров класса Pentium и последующих модификаций не используется потребителями, поскольку эти возможности в документации только упоминаются, но не рассматриваются.

Линейная адресация данных в реальном режиме DOS

В литературе по программированию описано три режима работы микропроцессоров серии 80x86: реальный режим (режим совместимости с архитектурой 8086), защищенный режим и режим виртуальных процессоров 8086 (являющийся подвидом защищенного режима).

Основной недостаток реального режима состоит в том, что адресное пространство имеет размер всего в 1 Мбайт и при этом сегментировано — «нарезано» на кусочки размером по 64 Кбайт. Одного мегабайта очень мало для современных ресурсоемких прикладных программ (текстовых и графических редакторов, геоинформационных

систем, систем проектирования и т. д.), а сегментация не позволяет нормально работать с видеопамятью и большими массивами данных.

Что можно сказать о защищенном и виртуальном режимах? Многие книги и учебники по микропроцессорам Intel *заканчиваются* главой «Переход в защищенный режим». Недостаток этого режима — необходимость *заново* создавать программное обеспечение для работы с периферийными устройствами на низком уровне, то есть фактически полностью переписывать *все* основные функции DOS. Можно, конечно, использовать Windows, но эта операционная система предназначена для офисных целей и плохо адаптируется к решению задач оперативного управления техническими системами. Кроме того, Windows забирает для собственных нужд изрядную часть ресурсов компьютера и ограничивает доступ к периферийным устройствам.

В некоторых случаях универсальные многозадачные операционные системы типа Windows и Unix неприменимы по причинам, не относящимся напрямую к области вычислительной техники. Первая причина — лицензионные соглашения между изготовителями и потребителями программ. Прочтите внимательно любую лицензию: разработчик программы не несет ответственности *ни за что*! Следовательно, за все сбои и неисправности расплачивается потребитель. Например, за аварию в системе управления транспортом разработчикам этой системы придется отвечать по статьям Уголовного кодекса. Что касается систем военного назначения, то вообще сомнительно, что на таких лицензионных условиях какая-либо программа может быть официально принята в эксплуатацию на территории России.

Вторая причина — огромный объем универсальных операционных систем: десятки миллионов строк на языках высокого уровня! Полностью протестировать такие системы невозможно — у фирмы Microsoft, например, хватает сил только на доскональную проверку небольшого ядра Windows! Тем более на это не способен потребитель, у которого нет всей документации. Даже в случае открытой системы типа Linux, если документация есть и все исходные коды доступны — попробуйте *доказать* военным или банкирам, что в системе нет скрытых ловушек и «черного хода»!

Создать собственную программу для переключения в защищенный режим и работы в нем — непростая задача. При работе с аппаратурой в защищенном режиме программист должен четко понимать,

какими возможностями аппаратуры пользоваться опасно. Например, приводимые в учебниках образцы программ для защищенного режима часто проявляют несовместимость с определенными конфигурациями оборудования, поскольку их авторы не имели достаточно широкой лабораторной базы для тестирования. Дело в том, что периферийные устройства всегда имеют какие-нибудь нестандартные особенности, добавляемые их изготовителями в рекламных целях. При работе в реальном режиме DOS такие особенности не применяются и потому никак не проявляются. Однако они могут показать себя с самой неприятной стороны при переключении в защищенный режим, когда программисту приходится перенастраивать периферийные устройства на новую модель организации оперативной памяти, перезаписывая при этом множество различных регистров аппаратуры. Возможны две ситуации: либо в стандартных регистрах некоторые разряды применяются нестандартным образом, но программисту об этом ничего не известно, либо вообще имеются какие-либо дополнительные регистры, не описанные в документации, но влияющие на режим работы системы. Возникает абсурдная ситуация: простой (реальный) режим работы задается процедурами BIOS фирмы-изготовителя системной платы, которая обычно хорошо осведомлена об особенностях применяемого на этой плате чипсета, а программы для перехода в защищенный режим вынуждены писать совершенно посторонние люди, не располагающие документацией в полном объеме. В BIOS включено некоторое количество процедур для работы в защищенном режиме, но они охватывают лишь часть необходимых операций.

Вообще говоря, изобилие управляющих регистров в современных персональных компьютерах (их общее количество достигает нескольких тысяч) — явление совершенно ненормальное, теоретически приводящее к увеличению количества возможных режимов работы до бесконечности. Поскольку протестировать функционирование системы в миллиардах различных режимов технически невозможно, разработчики программного обеспечения не могут использовать дополнительные средства и ограничены несколькими общепринятыми (стандартными) режимами. Чтобы убедиться в этом, достаточно сравнить полный набор команд любого периферийного устройства с реально используемым (например, в BIOS) подмножеством команд данного набора. Большая часть регистров в настоящее время в принципе не нужна: установкой режима работы периферийного устройства должен заниматься его встроенный специализированный процессор, а не центральный процессор компьютера.

Однако переход на новые технологии произойдет, вероятно, только после очередного кризиса в развитии компьютерной индустрии, а пока что приходится приспосабливаться к сложившейся ситуации.

Изложенные выше причины приводят к тому, что программисты вынуждены искать различные обходные пути. Один из возможных приемов — использование линейной адресации памяти. Линейная адресация — это наиболее простой, с точки зрения программиста, способ работы непосредственно с аппаратурой компьютера (логические адреса при этом совпадают с физическими). Различия в организации памяти в реальном, защищенном и линейном режимах работы процессора иллюстрирует рис. 2.1.

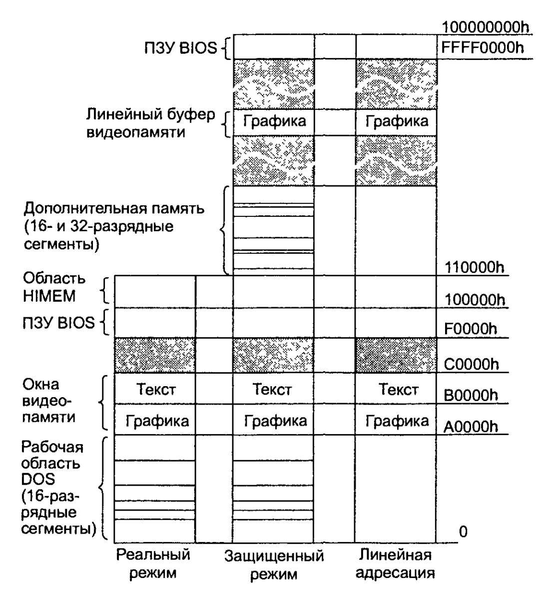


Рис. 2.1. Организация адресного пространства памяти в реальном, защищенном и линейном режимах работы процессора x86

Линейную адресацию можно использовать в специализированных программах, активно эксплуатирующих ресурсы компьютера — как

в играх, так и в системах автоматики, измерительных системах, системах управления, связи и т. п. Применение линейной адресации целесообразно в том случае, если проектируемая система предназначена для выполнения ограниченного, заранее известного набора функций и требует высокого быстродействия и надежности.

Разработчики процессоров начали внедрять линейную адресацию (в качестве одного из возможных режимов работы) при переходе с 16-разрядной архитектуры на 32-разрядную. Фирма Intel ввела такой режим в процессоре 80386, после чего он стал фактически стандартным (поддерживается не только всеми последующими моделями, но и всеми клонами архитектуры x86), однако остался недокументированным (почти не описан в литературе и не рассматривается в фирменном руководстве по программированию).

Для пользователей обычных персональных компьютеров линейная адресация в чистом виде интереса не представляет по тем же причинам, что и защищенный режим: DOS и BIOS функционируют только в реальном режиме с 64-килобайтными сегментами, и при переходе в любой другой режим программист оказывается один на один с аппаратурой компьютера — без документации. Однако кроме чистых режимов, процессоры Intel способны работать и в режимах гибридных.

Еще в 1989 году Томас Роден (Thomas Roden) предложил использовать интересную комбинацию сегментной (для кода и данных) и линейной (только для данных) адресации [90]. Предложенный им метод позволяет, находясь в обычном режиме DOS, работать со всей доступной памятью в пределах четырехгигабайтного адресного пространства процессора Intel 80386. Чтобы включить режим линейной адресации данных, необходимо снять ограничения на размер сегмента в теневом регистре, соответствующем одному из дополнительных сегментных регистров FS или GS (при необходимости описание архитектуры процессора Pentium вы можете найти в документации [66–68], размещенной в Интернете на сервере Intel для разработчиков). Через избранный регистр можно обращаться к любой области памяти с помощью прямой адресации или используя в качестве индексного любой 32-разрядный регистр общего назначения.

После снятия ограничения запись в выделенный для линейной адресации сегментный регистр выполнять нельзя, иначе нарушится информация в соответствующем ему теневом регистре (предел сегмента сохранится, но начальный адрес будет перезаписан новым значением). Однако стандартные компиляторы и функции

DOS с регистрами FS и GS не работают, и, соответственно, при вызове процедур эти регистры можно вообще «не трогать» — их не нужно сохранять и восстанавливать. Достаточно один раз снять ограничение на размер адресного пространства, и после выхода из программы (до перезагрузки компьютера) линейную адресацию можно будет использовать из любой другой программы DOS, как поступил в своем примере Томас Роден.

Рассмотрим более подробно процедуру переключения одного из дополнительных сегментных регистров в режим линейной адресации. Каждый сегментный регистр, как указано в документации [68], состоит из видимой и невидимой (теневой) частей. Информацию в видимую часть можно записывать напрямую при помощи обычных команд пересылки данных (MOV и т. д.), а для записи в невидимую часть применяются специальные команды, которые доступны только в защищенном режиме. Теневая часть представляет собой так называемый дескриптор (описатель) сегмента, длина которого равна 64 разрядам.

При переходе от 16-разрядной архитектуры к 32-разрядной (то есть от i286 к i386) разработчики нового процессора попытались сохранить совместимость снизу вверх по структуре системных регистров, в результате чего дескрипторы сегментов приобрели довольно уродливый (с точки зрения технической эстетики) вид — поля предела и базового адреса разделены на несколько частей. Кроме того, поле предела оказалось ограничено 20 разрядами, что вынудило разработчиков применить еще один радиолюбительский трюк — ввести бит гранулярности G, чтобы можно было задавать размер сегмента, превышающий 16 Мбайт.

Старшее 32-разрядное слово дескриптора сегмента

Базовый адрес 31–24	G	D/B	O	AVL	Предел сегмента 19–16	P	DPL	S	Тип	Базовый адрес 23–16
31 24	23	22	21	20	19 16	15	14 13	12	11 8	7 0

Младшее 32-разрядное слово дескриптора сегмента

Базовый адрес 15–0	Предел сегмента 15–0
31 16	15 0

Рис. 2.2. Формат дескриптора сегмента

Формат дескриптора сегмента показан на рис. 2.2. Дескриптор состоит из перечисленных ниже полей.

- Базовый адрес — 32-разрядное поле, задающее начальный адрес сегмента (в линейном адресном пространстве).
- Предел сегмента — 20-разрядное поле, которое определяет размер сегмента в байтах или 4-килобайтных страницах (в зависимости от значения бита гранулярности G). Поле предела содержит значение, которое должно быть на единицу меньше реального размера сегмента в байтах или страницах.
- Тип — 4-разрядное поле, определяющее тип сегмента и типы операций, которые допустимо с ним выполнять.
- Бит S — признак системного объекта (0 — дескриптор описывает системный объект, 1 — назначение сегмента описывается полем типа).
- DPL — 2-разрядное поле, определяющее уровень привилегий описываемого дескриптором сегмента.
- Бит P — признак присутствия сегмента в оперативной памяти компьютера (0 — сегмент «сброшен» на диск, 1 — сегмент присутствует в оперативной памяти).
- Бит AVL — свободный (available) бит, который может использоваться по усмотрению системного программиста.
- Бит D — признак используемого по умолчанию режима адресации данных (0 — 16-разрядная адресация, 1 — 32-разрядная).
- Бит G — гранулярность сегмента (0 — поле предела задает размер сегмента в байтах, 1 — в 4-килобайтных страницах).

В нашем случае признак используемого по умолчанию режима адресации данных D можно установить в 0 (использовать по умолчанию 16-разрядные операнды), но особой роли его значение не играет — в смешанном режиме сегментно-линейной адресации при работе с линейным сегментом строковые команды, использующие значение этого разряда, применять нельзя. Бит гранулярности G должен быть установлен в 1, чтобы обеспечить охват всего адресного пространства процессора.

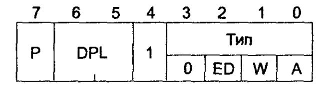


Рис. 2.3. Формат байта прав доступа для сегмента данных

Для сегментов данных формат байта прав доступа (включающего поле типа сегмента) имеет вид, показанный на рис. 2.3. Как видно

из рисунка, поле S для сегментов данных должно быть установлено в 1, а старший разряд поля типа должен иметь значение 0. Поля P и DPL уже упоминались выше. Бит присутствия сегмента P следует установить в 1 (сегмент присутствует в памяти), а в поле DPL нужно установить максимальный уровень привилегий (значение 00). Бит расширения вниз ED для сегментов данных имеет значение 0 (в отличие от стековых сегментов, для которых ED=1). Бит разрешения записи W следует установить в единицу, чтобы можно было не только считывать, но и записывать информацию в сегмент. Бит A фиксирует обращение к сегменту и автоматически устанавливается в единицу всякий раз, когда процессор производит операции считывания или записи с сегментом, описываемым данным дескриптором. При инициализации регистра бит A можно сбросить в 0.

PG	CD	NW	Зарезервировано	PG		NW	Зарезервировано	NE	ET	TS	EM	MP	PE

31 30 29 28 19 18 17 16 15 6 5 4 3 2 1 0

Рис. 2.4. Формат регистра управления CR0

Осуществить загрузку теневых регистров можно только в защищенном режиме. Для переключения режимов работы процессора используется регистр управления CR0, формат которого показан на рис. 2.4. Регистр CR0 содержит флаги, отражающие состояние процессора и управляющие режимами его работы. Назначение флагов следующее:

- PE (Protect Enable) — включение защищенного режима (0 — процессор работает в реальном режиме, 1 — в защищенном режиме);
- MP (Math Present) — признак наличия математического сопроцессора (0 — нет сопроцессора, 1 — сопроцессор присутствует в системе);
- EM (Emulation) — эмуляция математического сопроцессора (0 — выключена, 1 — включена);
- TS (Task Switched) — признак переключения задачи (флаг устанавливается в 1 при каждом переключении задач и проверяется перед выполнением команд математического сопроцессора);
- ET (Extension Type) — поддержка набора инструкций математического сопроцессора (0 — выключена, 1 — включена). В процессорах P6 флаг всегда установлен в 1;
- NE (Numeric Error) — встроенный механизм контроля ошибок математического сопроцессора (0 — выключен, 1 — включен);

- WP (Write Protect) — защита от записи информации в страницы уровня пользователя из процедур супервизора (0 — выключена, 1 — включена);
- AM (Alignment Mask) — автоматический контроль выравнивания (0 — запрещен, 1 — разрешен);
- NW (Not Write-through) — запрещение сквозной записи (0 — сквозная запись разрешена, 1 — запрещена);
- CD (Cache Disable) — запрещение кэш-памяти (0 — использование кэш-памяти разрешено, 1 — запрещено);
- PG (Paging) — страничное преобразование (0 — запрещено, 1 — разрешено).

Набор подпрограмм, необходимых для переключения сегментного регистра GS в режим линейной адресации, показан в листинге 2.1. Как указано выше, перезапись содержимого теневого регистра процессора возможна только в защищенном режиме, а переход в этот режим, как видно из листинга, требует ряда дополнительных операций, выполняемых процедурой Initialization. В частности, нужно перенастроить на специально выделенные в кодовом сегменте области данных регистры DS, SS и SP. В момент перенастройки регистров стека должны быть запрещены прерывания, поскольку некоторые обработчики прерываний пишут информацию в стек прерываемой программы.

Процедура SetLAddrModeForGS, непосредственно осуществляющая перенастройку регистра GS в режим линейной адресации, воспроизводит (с незначительными изменениями) метод Родена. Прежде чем осуществить переключение, нужно подготовить таблицу GDT (настроить на текущие сегменты кода и данных) и загрузить ее. Затем нужно войти в защищенный режим — установить в единицу бит PE регистра CR0, а остальные разряды сохранить без изменений (в том виде, в котором они находились при работе в реальном режиме). В защищенном режиме нужно перезагрузить сегментные регистры, сняв при этом ограничения с GS, и сразу же вернуться в реальный режим DOS, сбросив в ноль бит PE. Длительное пребывание в защищенном режиме нежелательно, поскольку переключение в него выполнялось по упрощенной схеме: таблица прерываний не создавалась, а сами прерывания были просто отключены.

После выполнения процедуры SetLAddrModeForGS обязательно следует отменить заворачивание адресного пространства, то есть разблокировать адресную линию A20, которая управляется контроллером

клавиатуры. Для этого необходимо послать в порт А контроллера соответствующую команду по правилам, описанным в главе 1 «Работа с клавиатурой». Посылка команды осуществляется при помощи процедур Enable_A20 и Wait8042BufferEmpty.

Листинг 2.1. Подпрограмма, устанавливающая режим линейной адресации данных

```
; Порт, управляющий запретом немаскируемых прерываний
CMOS_ADDR       equ  0070h
CMOS_DATA       equ  0071h
; Селекторы сегментов
SYS_PROT_CS     equ  0008h
SYS_REAL_SEG    equ  0010h
SYS_MONDO_SEG   equ  0018h

CODESEG
;**********************************************
;* ВКЛЮЧЕНИЕ РЕЖИМА ЛИНЕЙНОЙ АДРЕСАЦИИ ПАМЯТИ *
;*       (процедура параметров не имеет)       *
;**********************************************
PROC Initialization NEAR
        pushad
; Сохранить значения сегментных регистров в
; реальном режиме (кроме GS)
        mov     [CS:Save_SP],SP
        mov     AX,SS
        mov     [CS:Save_SS],AX
        mov     AX,DS
        mov     [CS:Save_DS],AX
; (работаем теперь только с кодовым сегментом)
        mov     AX,CS
        mov     [word ptr CS:Self_Mod_CS],AX
        mov     DS,AX
        cli
        mov     SS,AX
        mov     SP,offset Local_Stk_Top
        sti

; Установить режим линейной адресации
        call    SetLAddrModeForGS

; Восстановить значения сегментных регистров
        cli
        mov     SP,[CS:Save_SP]
        mov     AX,[CS:Save_SS]
        mov     SS,AX
        mov     AX,[CS:Save_DS]
        mov     DS,AX
        sti
```

```
; Разрешить работу линии A20
        call    Enable_A20
        popad
        ret
ENDP Initialization

; Область сохранения значений сегментных регистров
Save_SP DW ?
Save_SS DW ?
Save_DS DW ?
; Указатель на GDT
GDTPtr  DQ ?
; Таблица дескрипторов сегментов для
; входа в защищенный режим
GDT DW 00000h,00000h,00000h,00000h ;не используется
    DW 0FFFFh,00000h,09A00h,00000h ;сегмент кода CS
    DW 0FFFFh,00000h,09200h,00000h ;сегмент данных DS
    DW 0FFFFh,00000h,09200h,0008Fh ;сегмент GS
; Локальный стек для защищенного режима
; (организован внутри кодового сегмента)
label GDTEnd word
        DB 255 DUP(0FFh)
Local_Stk_Top DB (0FFh)

;*******************************************
;*        ОТМЕНИТЬ ПРЕДЕЛ СЕГМЕНТА GS      *
;* Процедура изменяет содержимое теневого  *
;* регистра GS таким образом, что становится *
;* возможной линейная адресация через него *
;* 4 Gb памяти в реальном режиме           *
;*******************************************
PROC SetLAddrModeForGS near
; Вычислить линейный адрес кодового сегмента
        mov     AX,CS
        movzx   EAX,AX
        shl     EAX,4   ;умножить номер параграфа на 16
        mov     EBX,EAX ;сохранить линейный адрес в EBX
; Занести младшее слово линейного адреса в дескрипторы
; сегментов кода и данных
        mov     [word ptr CS:GDT+10],AX
        mov     [word ptr CS:GDT+18],AX
        ; Переставить местами старшее и младшее слова
        ror     EAX,16
; Занести биты 16-23 линейного адреса в дескрипторы
; сегментов кода и данных
        mov     [byte ptr CS:GDT+12],AL
        mov     [byte ptr CS:GDT+20],AL
; Установить предел (Limit) и базу (Base) для GDTR
        add     EBX, offset GDT
```

продолжение ⇨

Листинг 2.1 (*продолжение*)

```
        mov [word ptr CS:GDTPtr],(offset GDTEnd-GDT-1)
        mov     [dword ptr CS:GDTPtr+2],EBX
; Сохранить регистр флагов
        pushf
; Запретить прерывания, так как таблица прерываний IDT
; не сформирована для защищенного режима
        cli
; Запретить немаскируемые прерывания NMI
        in      AL,CMOS_ADDR
        mov     AH,AL
        or      AL,080h       ;установить старший разряд
        out     CMOS_ADDR,AL ;не затрагивая остальные
        and     AH,080h
        ; Запомнить старое состояние маски NMI
        mov     CH,AH
; Перейти в защищенный режим
        lgdt    [fword ptr CS:GDTPtr]
        mov     BX,CS    ;запомнить сегмент кода
        mov     EAX,CR0
        or      AL,01b   ;установить бит PE
        mov     CR0,EAX  ;защита разрешена
        ; Безусловный дальний переход на метку SetPMode
        ; (очистить очередь команд и перезагрузить CS)
                DB      0EAh
                DW      (offset SetPMode)
                DW      SYS_PROT_CS
SetPMode:
        ; Подготовить границы сегментов
        mov     AX,SYS_REAL_SEG
        mov     SS,AX
        mov     DS,AX
        mov     ES,AX
        mov     FS,AX
        ; Снять ограничения с сегмента GS
        mov     AX,SYS_MONDO_SEG
        mov     GS,AX
; Вернуться в реальный режим
        mov     EAX,CR0
        and     AL,11111110b ;сбросить бит PE
        mov     CR0,EAX      ;защита отключена

        ; Безусловный дальний переход на метку SetRMode
        ; (очистить очередь команд и перезагрузить CS)
           DB 0EAh
           DW (offset SetRMode)
Self_Mod_CS DW ?

SetRMode:
        ; Регистры стека и данных
```

```
        ; настроить на сегмент кода
        mov     SS,BX
        mov     DS,BX
        ; Обнулить дополнительные сегментные
        ; регистры данных (GS не трогать!)
        xor     AX,AX
        mov     ES,AX
        mov     FS,AX
        ; Возврат в реальный режим,
        ; прерывания снова разрешены
        in      AL,CMOS_ADDR
        and     AL,07Fh
        or      AL,CH
        out     CMOS_ADDR,AL
        popf
        ret
ENDP SetLAddrModeForGS

;***************************************
;* Разрешить работу с памятью выше 1 Мб *
;***************************************
PROC Enable_A20 near
        call    Wait8042BufferEmpty
        mov     AL,0D1h ;команда управления линий A20
        out     64h,AL
        call    Wait8042BufferEmpty
        mov     AL,0DFh ;разрешить работу линии
        out     60h,AL
        call    Wait8042BufferEmpty
        ret
ENDP Enable_A20

;*********************************************
;*   ОЖИДАНИЕ ОЧИСТКИ ВХОДНОГО БУФЕРА I8042    *
;* При выходе из процедуры:                  *
;* флаг ZF установлен - нормальное завершение, *
;* флаг ZF сброшен - ошибка тайм-аута.       *
;*********************************************
proc Wait8042BufferEmpty near
        push    CX
        mov     CX,0FFFFh  ;задать число циклов
@@kb:   in      AL,64h     ;получить статус
        test    AL,10b     ;буфер i8042 свободен?
        loopnz  @@kb       ;если нет, то цикл
        pop     CX
        ; (если при выходе сброшен флаг ZF - ошибка)
        ret
endp Wait8042BufferEmpty
ENDS
```

ВНИМАНИЕ

Как уже было указано выше, после выхода из защищенного режима нельзя перезаписывать регистр GS, иначе будет полностью или частично стерта информация в соответствующем теневом регистре. В частности, нельзя выполнять операции сохранения/восстановления содержимого регистра при помощи команд работы со стеком push и pop.

При использовании нестандартных режимов работы возникают определенные трудности в процессе отладки программ: стандартные программы-отладчики становятся неудобными. Во многих случаях, однако, достаточно использовать простую отладочную печать. В листинге 2.2 приведена подпрограмма ShowRegs, отображающая на экран содержимое регистров общего назначения, сегментных регистров, регистра флагов и регистра CR0. Недостаток этого упрощенного примера заключается в том, что ShowRegs *не сохраняет содержимое видеопамяти*. Однако при использовании линейной адресации программу нетрудно усовершенствовать, если есть достаточный запас оперативной памяти: в текстовом режиме для сохранения одной страницы нужно менее 4 Кбайт, а в графическом режиме TrueColor32 с разрешением 1920×1280 требуется уже 9,5 Мбайт.

Листинг 2.2. Отладочная подпрограмма, предназначенная для отображения на экран содержимого регистров процессора

```
DATASEG
label REGROW_386 byte
   DB 0,0,'EAX =',0
   DB 1,0,'EBX =',0
   DB 2,0,'ECX =',0
   DB 3,0,'EDX =',0
   DB 4,0,'ESI =',0
   DB 5,0,'EDI =',0
   DB 6,0,'EBP =',0
   DB 7,0,'ESP =',0
   DB B,0,'IP  =',0
   DB 9,0,'CS  =',0
   DB 10,0,'DS  =',0
   DB 11,0,'ES  =',0
   DB 12,0,'FS  =',0
   DB 13,0,'GS  =',0
   DB 14,0,'SS  =',0
   DB 16,8,'              AVR  NIOODIT SZ A P C',0
   DB 17,B,'              CMF  TPLFFFF FF F F F',0
   DB 1B,0,'Флаги:',0
   DB 20,B,'PCN           A V             NETEMP',0
```

```
   DB 21,8,'GDW          M P           ETSMPE',0
   DB 22,0,'CR0:',0
   DB 24,15
   DB 'Для продолжения работы нажмите любую клавишу',0

CODESEG
;********************************************
;* ВЫВЕСТИ НА ЭКРАН ДАМП РЕГИСТРОВ ПРОЦЕССОРА *
;*     (процедура параметров не имеет)      *
;********************************************
PROC ShowRegs FAR
        pushad
        pushfd
        push    DS
        mov     BP,SP
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
; Сохраняем глобальные переменные
        mov     AL,[TextColorAndBackground]
        push    AX
        push    [ScreenString]
        push    [ScreenColumn]
; Очищаем экран
        call    ClearScreen
; Вывести 21 строку текста
        mov     [TextColorAndBackground],YELLOW
        mov     SI, offset REGROW_386
        mov     CX,22
@@GLB:  call    ShowString
        loop    @@GLB
; Вывести содержимое регистров
        mov     [TextColorAndBackground],WHITE
        MShowHexDWord  0,6,[BP+34] ;Показать EAX
        MShowHexDWord  1,6,[BP+22] ;Показать EBX
        MShowHexDWord  2,6,[BP+30] ;Показать ECX
        MShowHexDWord  3,6,[BP+26] ;Показать EDX
        MShowHexDWord  4,6,[BP+10] ;Показать ESI
        MShowHexDWord  5,6,[BP+6]  ;Показать EDI
        MShowHexDWord  6,6,[BP+14] ;Показать EBP
        MShowHexDWord  7,6,[BP+18] ;Показать ESP
        MShowHexWord   8,6,[BP+38] ;Показать IP
        MShowHexWord   9,6,[BP+40] ;Показать CS
        MShowHexWord  10,6,[BP]    ;Показать DS
        MShowHexWord  11,6,ES      ;Показать ES
        MShowHexWord  12,6,FS      ;Показать FS
        MShowHexWord  13,6,GS      ;Показать GS
        MShowHexWord  14,6,SS      ;Показать SS
        MShowBinDWord 18,8,[BP+2]
        MShowBinDWord 22,8,CR0     ;Показать CR0
```

продолжение

Листинг 2.2 (*продолжение*)

```
; Ожидаем нажатия любого символа на клавиатуре
        call    GetChar
; Очищаем экран
        call    ClearScreen

; Восстановить глобальные переменные
        pop     [ScreenColumn]
        pop     [ScreenString]
        pop     AX
        mov     [TextColorAndBackground],AL
        pop     DS
        popfd
        popad
        ret
ENDP ShowRegs
ENDS
```

В программе LAddrTest, показанной в листинге 2.3, используются процедуры из листингов 2.1 и 2.2 для включения режима линейной адресации и демонстрации изменения содержимого сегментных регистров, которое при этом происходит (процедура установки линейного режима перезаписывает теневой регистр у регистра GS, а регистры ES и FS просто обнуляет). После выполнения программы режим линейной адресации данных *сохраняется*, и любая другая программа, в том числе написанная на языке высокого уровня, может через GS обращаться к любой области памяти по физическому адресу.

Листинг 2.3. Включение режима линейной адресации

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Текстовые сообщения
Text1 DB 0,19,"Включение режима "
      DB "линейной адресации данных",0
```

```
      DB 11,0,"Для просмотра "
      DB "содержимого регистров процессора",0
      DB 12,0,"перед запуском процедуры "
      DB "перехода в режим",0
      DB 13,0,"линейной адресации нажмите "
      DB "любую клавишу.",0
Text2 DB 11,0,"Произведено переключение в "
      DB "режим линейной адресации.",0
      DB 12,0,"Для просмотра содержимого "
      DB "регистров процессора",0
      DB 13,0,"нажмите любую клавишу.",0
Text3 DB 11,0,"После завершения данной "
      DB "программы регистр GS",0
      DB 12,0,"может использовать для "
      DB "линейной адресации",0
      DB 13,0,"любая другая программа.",0
      DB 24,1B,"Для выхода из программы "
      DB "нажмите любую клавишу.",0
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC LAddrTest
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести первое текстовое сообщение
; на экран зеленым цветом
        mov     [TextColorAndBackground],LIGHTGREEN
        MShowText 4,Text1
        ; Ожидать нажатия любой клавиши
        call    GetChar
; Занести контрольное число в дополнительные
; сегментные регистры данных
        mov     AX,0ABCDh
        mov     ES,AX
        mov     FS,AX
        mov     GS,AX
; Показать содержимое регистров процессора
        call    far ShowRegs
```

продолжение

Листинг 2.3 (*продолжение*)

```
; Установить режим прямой адресации памяти
        call    Initialization
; Вывести второе текстовое сообщение
; на экран голубым цветом
        mov     [TextColorAndBackground],LIGHTCYAN
        MShowText 3,Text2
        ; Ожидать нажатия любой клавиши
        call    GetChar
; Показать содержимое регистров процессора
        call    far ShowRegs

; Вывести третье текстовое сообщение
; на экран желтым цветом
        mov     [TextColorAndBackground],YELLOW
        MShowText 4,Text3
        ; Ожидать нажатия любой клавиши
        call    GetChar
; Установить текстовый режим
        mov     ax,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP LAddrTest
ENDS
; Подключить набор процедур вывода/вывода данных
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить подпрограмму, отображающую на экране
; содержимое регистров процессора
include "list2_02.inc"

END
```

Листинг 2.4 демонстрирует использование линейной адресации для отображения содержимого памяти компьютера на экран, то есть выдачи дампа памяти. Программа MemoryDump позволяет просматривать все адресное пространство, а не только оперативную память. Можно, например, считывать память видеоконтроллера или вообще неиспользуемые области.

Кроме процедур ввода/вывода общего назначения, в MemoryDump используются также следующие подпрограммы:

- процедура HexToBin32 осуществляет перевод числа (введенного с клавиатуры адреса) из шестнадцатеричного кода в двоичный;

- процедура GetAddressOrCommand принимает команды, вводимые с клавиатуры (введенное число воспринимается как линейный адрес памяти в шестнадцатеричном коде, нажатие на управляющие клавиши — как команда).

Листинг 2.4. Использование линейной адресации для вывода на экран содержимого оперативной памяти

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Текстовые сообщения
Txt1 DB LIGHTMAGENTA,0,28,"Дамп оперативной памяти",0
     DB YELLOW,2,0,"Адрес:",0
     DB LIGHTGREEN,2,11
     DB "Шестнадцатеричное представление:",0
     DB LIGHTCYAN,2,61,"ASCII-коды:",0
     DB LIGHTRED,21,0,"Введите число "
     DB "или нажмите управляющую клавишу:",0
Txt2 DB 23,0, "Стрелка вниз - следующие 256 байт;",0
     DB 23,35, "Стрелка вверх - предыдущие 256 байт;",0
     DB 24,0, "Enter - завершение ввода адреса;",0
     DB 24,33, "Esc - отмена ввода адреса;",0
     DB 24,60, "F10 - выход.",0
; Количество введенных символов числа
CharacterCounter DB 0
; Позиция для ввода адреса на экране
OutAddress DB 21,47
; Строка для ввода адреса
AddressString DB 9 DUP(0)
; Строка пробелов для "затирания" числа
SpaceString DB 21,47,9 DUP(' '),0
; Начальный адрес
StartAddress DD 0
```

продолжение ⇗

Листинг 2.4 (*продолжение*)

```
; Код команды
CommandByte DB 0
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC MemoryDump
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Устанавливаем режим прямой адресации памяти
        call    Initialization
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 5,Txt1
        mov     [TextColorAndBackground],WHITE
        MShowText 5,Txt2
; Установить белый цвет символов и черный фон
        mov     [TextColorAndBackground],WHITE
; Отобразить символы-разделители колонок
        mov     AL,0B3h
        mov     [ScreenString],2
        mov     [ScreenColumn],9
        call    ShowASCIIChar
        mov     [ScreenColumn],59
        call    ShowASCIIChar
        mov     [ScreenString],3
        mov     [ScreenColumn],9
        call    ShowASCIIChar
        mov     [ScreenColumn],59
        call    ShowASCIIChar

; Инициализируем переменные
        mov     [StartAddress],0
        mov     [CommandByte],0
; ВНЕШНИЙ ЦИКЛ
@@q0:   mov     EBX,[StartAddress]
        mov     [ScreenString],4
        mov     DX,16
@@q1:   mov     [ScreenColumn],0
```

```
; Отобразить линейный адрес первого байта в группе
        mov     [TextColorAndBackground],YELLOW
        mov     EAX,EBX
        call    ShowHexDWord
; Отобразить символ-разделитель колонок
        mov     [TextColorAndBackground],WHITE
        inc     [ScreenColumn]
        mov     AL,0B3h
        call    ShowASCIIChar
        inc     [ScreenColumn]
; Отобразить очередную группу байтов
; в шестнадцатеричном коде
        mov     CX,16
        mov     [TextColorAndBackground],LIGHTGREEN
@@q2:   mov     AL,[GS:EBX]
        call    ShowHexByte
        inc     [ScreenColumn]
        inc     EBX
        loop    @@q2
; Отобразить символ-разделитель колонок
        mov     [TextColorAndBackground],WHITE
        mov     AL,0B3h
        call    ShowASCIIChar
        inc     [ScreenColumn]
        ; Вернуться назад на 16 символов
        sub     EBX,16
; Отобразить очередную группу байтов в кодах ASCII
        mov     CX,16
        mov     [TextColorAndBackground],LIGHTCYAN
@@q3:   mov     AL,[GS:EBX]
        call    ShowASCIIChar
        inc     EBX
        loop    @@q3
        inc     [ScreenString]
        dec     DX
        jnz     @@q1

        ; Ожидать нажатия любой клавиши
        call    GetAddressOrCommand
        cmp     [CommandByte],F10
        jne     @@q0

@@End:  ; Установить текстовый режим
        mov     ax,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP MemoryDump
```

продолжение ⇒

Листинг 2.4 (*продолжение*)

```
;****************************************************
;* ПЕРЕВОД ЧИСЛА ИЗ ШЕСТНАДЦАТЕРИЧНОГО КОДА В ДВОИЧНЫЙ *
;* DS:SI - число в коде ASCII.                      *
;* Результат возвращается в EAX.                    *
;****************************************************
PROC HexToBin32 near
        push    EBX
        push    CX
        push    SI
        cld
        xor     EBX,EBX ;обнуляем накопитель
        xor     CX,CX   ;обнуляем счетчик цифр
@@h0:   lodsb
        ; Проверка на ноль (признак конца строки)
        and     AL,AL
        jz      @@h4
        ; Проверка на диапазон '0'-'9'
        cmp     AL,'0'
        jb      @@Error
        cmp     AL,'9'
        ja      @@h1
        sub     AL,'0'
        jmp short @@h3
@@h1:   ; Проверка на диапазон 'A'-'F'
        cmp     AL,'A'
        jb      @@Error
        cmp     AL,'F'
        ja      @@h2
        sub     AL,'A'-10
        jmp short @@h3
@@h2:   ; Проверка на диапазон 'a'-'f'
        cmp     AL,'a'
        jb      @@Error
        cmp     AL,'f'
        ja      @@Error
        sub     AL,'a'-10
@@h3:   ; Дописать к результату
        ; очередные 4 разряда справа
        shl     EBX,4
        or      BL,AL
        inc     CX
        cmp     CX,8
        jbe     @@h0
        ; Если в числе больше 8 цифр - ошибка
        jmp short @@Error
@@h4:   ; Успешное завершение - результат в EAX
        mov     EAX,EBX
        jmp short @@End
```

```
@@Error:: Ошибка - обнулить результат
        xor     EAX,EAX
@@End:  pop     SI
        pop     CX
        pop     EBX
        ret
ENDP HexToBin32

;**********************************************
;* ПРИНЯТЬ С КЛАВИАТУРЫ НОВЫЙ АДРЕС ИЛИ КОМАНДУ *
;**********************************************
PROC GetAddressOrCommand near
        pushad
        ; Использовать при выводе белый цвет, черный фон
        mov     [TextColorAndBackground],WHITE
        ; Установить номер строки поля ввода
        mov     [ScreenString],21
@@GetAddressOrCommand:
; Инициализировать переменные
        ; Обнулить счетчик цифр
        mov     [CharacterCounter],0
        ; Очистить строку
        mov     DI,offset AddressString
        mov     [byte ptr DS:DI],0
        ; Очистить позицию ввода (забить пробелаии)
        MShowString SpaceString
        ; Установить курсор в позицию ввода
        mov     [ScreenColumn],47
        mov     AL,[CharacterCounter]
        add     [byte ptr ScreenColumn],AL
        call    SetCursorPosition
        ; Ввести цифру или команду
        call    GetChar
        ; Адрес или команда?
        cmp     AL,0
        jz      @@Command
        ; Введена первая цифра числа

; ВВОД АДРЕСА В ШЕСТНАДЦАТЕРИЧНОМ КДДЕ
@@Address:
        ; Проверка на диапазон '0'-'9'
        cmp     AL,'0'
        jb      @@AddressError
        cmp     AL,'9'
        jbe     @@WriteChar
        ; Проверка на диапазон 'A'-'F'
        cmp     AL,'A'
        jb      @@AddressError
        cmp     AL,'F'
        jbe     @@WriteChar
```

продолжение ⇒

Листинг 2.4 (*продолжение*)

```
        ; Проверка на диапазон 'a'-'f'
        cmp     AL,'a'
        jb      @@AddressError
        cmp     AL,'f'
        ja      @@AddressError
@@WriteChar:
        ; Проверить количество цифр
        cmp     [CharacterCounter],8
        jae     @@AddressError
        inc     [CharacterCounter]
        ; Записать цифру в число
        mov     [DS:DI],AL
        inc     DI
        ; Передвинуть признак конца строки
        ; в следующий разряд
        mov     [byte ptr DS:DI],0
        ; Отобразить число на экране
        MShowString SpaceString
        MShowString OutAddress
@@GetNextChar:
        ; Отобразить курсор в новой позиции ввода
        mov     [ScreenColumn],47
        mov     AL,[CharacterCounter]
        add     [byte ptr ScreenColumn],AL
        call    SetCursorPosition
        ; Ожидать ввода следующего символа
        call    GetChar
        cmp     AL,0
        jne     @@Address

; Проанализировать код нажатой клавиши
        cmp     AH,B_Esc        ;отмена ввода адреса
        je      @@GetAddressOrCommand

@@TestF10:
        cmp     AH,F10          ;"Выход"
        jne     @@TestRubout
        mov     [CommandByte],AH
        jmp     @@End

@@TestRubout:
        cmp     AH,B_RUBOUT     ;"Забой"
        jne     @@TestEnter
        cmp     [CharacterCounter],0
        je      @@AddressError
        ; Передвинуть признак конца строки
        ; на разряд влево
```

```
        dec     DI
        dec     [CharacterCounter]
        mov     [byte ptr DS:DI],0
        ; Отобразить число на экране
        MShowString SpaceString
        MShowString OutAddress
        jmp     @@GetNextChar

@@TestEnter:
        cmp     AH,B_Enter      ;завершение ввода числа
        jne     @@AddressError
        mov     [CommandByte],AH
        mov     SI,offset AddressString
        call    HexToBin32
        mov     [StartAddress],EAX
        jmp short @@End

@@AddressError:
        call    Beep
        jmp     @@GetNextChar

; ОБРАБОТКА "КОМАНД"
@@Command:
        cmp     AH,F10          ;"Выход"
        jne     @@TestDn
        mov     [CommandByte],AH
        jmp short @@End

@@TestDn:
        cmp     AH,B_DN         ;"Стрелка вниз"
        jne     @@TestUp
        mov     [CommandByte],AH
        add     [StartAddress],256
        jmp short @@End
@@TestUp:
        cmp     AH,B_UP         ;"Стрелка вверх"
        jne     @@CommandError
        mov     [CommandByte],AH
        sub     [StartAddress],256
        jmp short @@End

@@CommandError:
        call    Beep
        jmp     @@GetAddressOrCommand
@@End:  popad
        ret
ENDP GetAddressOrCommand
ENDS
```

продолжение ⇨

Листинг 2.4 (*продолжение*)

```
; Подключить набор процедур вывода/вывода данных
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"

END
```

Я проверял метод Родена не только на процессорах Intel, но и на клонах, изготовленных AMD, Cyrix, IBM, TI. На всех протестированных компьютерах переход в режим линейной адресации данных проходил нормально, то есть метод не только работоспособен, но и универсален! Метод Родена в свое время не был оценен по достоинству, поскольку обычный объем памяти персональных компьютеров составлял тогда 1–2 Мбайт и преимущества линейной адресации не были очевидными. Резкое увеличение объема памяти в устройствах массового применения произошло гораздо позже — начиная с 1995 года. В это же время был внедрен новый стандарт на видеоконтроллеры (VESA 2.0) и появилась возможность линейной адресации видеопамяти, однако о методе Родена программисты уже успели напрочь забыть. Между тем, совместное использование линейной адресации данных в оперативной памяти и линейного пространства видеопамяти дает наибольший выигрыш по скорости выполнения программ и позволяет сильно упростить алгоритмы построения изображений.

Таким образом, метод Томаса Родена обладает следующими основными преимущества:

- имеется свободный доступ ко всем аппаратным ресурсам компьютера;
- возможна линейная адресация всей оперативной памяти и памяти видеоконтроллера;
- логические и физические адреса отображенной на шину процессора памяти периферийных устройств совпадают;
- метод совместим с клонами процессоров Intel;
- сохраняется возможность использования всех функций DOS и BIOS, как в обычном реальном режиме работы процессора.

Последнее свойство особенно важно: не нужно разрабатывать собственные программы для работы с периферийными устройствами на уровне регистров — следовательно, не проявляются и не создают лишних проблем нестандартные особенности оборудования.

Основной недостаток метода Родена — существенное ослабление защиты памяти. Поскольку отменен контроль за границей сегмента данных, работающая с линейным пространством подпрограмма в случае ошибки адресации или зацикливания может не только разрушить смежные данные, но и вообще стереть все содержимое оперативной памяти, в том числе все программы и резидентную часть операционной системы. Чаще всего стирается таблица векторов прерываний, размещенная в начале адресного пространства. Следовательно, необходимо ограничивать число подпрограмм, работающих с линейной адресацией, и очень тщательно их отлаживать.

Второй недостаток прямо вытекает из первого — работа в реальном режиме DOS и ослабление защиты не позволяют реализовать многозадачность. Однако для решения прикладных задач часто вполне достаточно фоново-оперативного режима работы, когда всеми ресурсами системы распоряжается один программный модуль, а остальные предназначены для узкоспециальных целей и вызываются на короткие промежутки времени через механизм прерываний. Иными словами, доступ к видеопамяти и всей оперативной памяти должен быть лишь у основной программы, а вспомогательные процедуры и драйверы периферийных устройств могут хранить свои данные только в основной области памяти DOS (то есть в пределах первого мегабайта адресного пространства). Линейная адресация сама по себе не накладывает слишком жестких ограничений на работу системы, поскольку персональные компьютеры вообще функционируют в основном в однозадачном режиме: аппаратные средства для реализации многозадачности имеются уже давно, но сильные ограничения накладывают физиологические и психологические особенности человека, который сидит за компьютером. Любая серьезная работа требует от оператора полной концентрации внимания на одном процессе. То же самое относится к компьютерным играм: невозможно одновременно играть в Quake и редактировать текст.

Третий недостаток: строковые команды процессора x86 в реальном режиме не пригодны для работы с сегментом, настроенным на линейную адресацию памяти. Это не очень существенный недостаток, поскольку внутренняя RISC-архитектура современных процессоров позволяет выполнять группу из нескольких простых команд с той же скоростью, что и одну сложную составную команду, выполняющую аналогичную операцию. Кроме того, процессор выполняет

внутренние операции быстрее, чем операции обращения к оперативной памяти, и гораздо быстрее, чем операции чтения/записи в видеопамять.

В целом можно сказать, что предложенный Роденом режим — это в первую очередь режим учебно-отладочный. Его очень удобно применять в процессе освоения методов непосредственной работы с периферийными устройствами. Во-первых, линейная адресация абсолютно прозрачна — область памяти устройства можно просматривать прямо по физическому адресу. Во-вторых, исследуемое устройство можно рассматривать изолированно, исключив опасность возникновения паразитных взаимодействий с другими аппаратными компонентами и посторонним программным обеспечением. Линейная адресация — это лестница, позволяющая осуществить постепенный переход от реального к защищенному режиму (рис. 2.5).

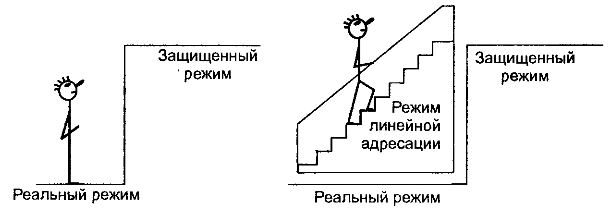


Рис. 2.5. Линейная адресация как промежуточный учебный режим

В компьютерных играх и других программах массового применения какая-либо защита памяти обязательно должна присутствовать. Поэтому для таких приложений вместо линейной адресации можно использовать сегментную, настроив один из дополнительных регистров на всю область расширенной памяти, а второй — на видеопамять (рис. 2.6). Способ настройки точно такой же, как при установке линейной адресации, но начальное смещение сегментов уже не нулевое, а размер должен совпадать с объемом дополнительной памяти и видеопамяти соответственно. Пример такой программы будет рассматриваться в главе 4 «Видеоконтроллеры», поскольку для настройки сегментного регистра на видеопамять надо определить, на какую область адресного пространства она была отображена в момент начальной загрузки операционной системы.

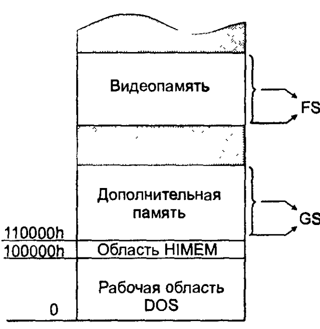


Рис. 2.6. Использование дополнительных сегментных регистров для адресации видеопамяти и дополнительной памяти

Перевод чисел из десятичного кода в двоичный и наоборот

Перевод чисел из одной системы счисления в другую выполняется при помощи математического сопроцессора очень просто — используется тот же механизм преобразования форматов данных, с помощью которого целые числа превращаются в вещественные и наоборот. Возможность применения этого способа имелась уже в самом первом сопроцессоре (i8087), однако об этом не упоминается ни в одном учебнике!

Математический сопроцессор понимает семь различных форматов представления чисел (параметры которых кратко описаны в табл. 2.1) и может выполнять любые преобразования из одного формата в другой, хотя при этом возможна потеря точности или переполнение разрядной сетки. Список команд передачи данных, которые можно использовать для преобразования чисел в различные форматы, приведен в табл. 2.2.

Таблица 2.1. Форматы данных математического сопроцессора

Тип	Размер, бит	Точность, бит	Диапазон
Короткое вещественное	32	24	От 2^{-129} до 2^{127}
Длинное вещественное	64	53	От 2^{-1022} до 2^{1023}

продолжение ⇨

Таблица 2.1 (*продолжение*)

Тип	Размер, бит	Точность, бит	Диапазон
Расширенное вещественное	80	64	От 2^{-16382} до 2^{16383}
Целое слово	16	15	От -2^{15} до $2^{15}-1$
Короткое целое	32	31	От -2^{31} до $2^{31}-1$
Длинное целое	64	63	От -2^{63} до $2^{63}-1$
Упакованное десятичное	80	18 дес. цифр	От $(-10^{18}+1)$ до $(10^{18}-1)$

Рассмотрим порядок перевода целого десятичного числа, записанного в ASCII-коде, в двоичный код.

1. Преобразовать число из ASCII-кода в упакованный десятичный формат. В случае наличия в числе посторонних знаков, не являющихся цифрами, выдать сообщение об ошибке.
2. Число в упакованном формате занести в стек сопроцессора.
3. Извлечь число из стека сопроцессора и сохранить его в памяти в заданном целочисленном или вещественном формате.

Таблица 2.2. Команды передачи данных математического сопроцессора

Команда	Выполняемая операция
FBLD	Преобразовать упакованное десятичное число в расширенный вещественный формат и занести в стек сопроцессора
FBSTP	Число, находящееся в вершине стека сопроцессора, сохранить в упакованном десятичном формате и извлечь из стека
FILD	Преобразовать целое число в расширенный вещественный формат и занести в стек сопроцессора
FIST	Число, находящееся в вершине стека сопроцессора, сохранить в заданном целочисленном формате
FISTP	Число, находящееся в вершине стека сопроцессора, сохранить в заданном целочисленном формате и извлечь из стека
FLD	Преобразовать вещественное число в расширенный формат и занести в стек сопроцессора
FST	Число, находящееся в вершине стека сопроцессора, сохранить в заданном вещественном формате
FSTP	Число, находящееся в вершине стека сопроцессора, сохранить в заданном вещественном формате и извлечь из стека

ПРИМЕЧАНИЕ

Целые числа в двоичном коде можно хранить как в целочисленных, так и в вещественных форматах.

Операция перевода целого числа из двоичного кода в десятичный выполняется в обратном порядке.

1. Число, хранящееся в одном из возможных двоичных форматов (целочисленном или вещественном), занести в стек сопроцессора при помощи соответствующей команды загрузки.
2. Извлечь число из стека сопроцессора в упакованном десятичном формате.
3. Преобразовать число из упакованного формата в ASCII-код.

При переводе вещественных чисел из десятичного формата в двоичный необходима дополнительная операция — масштабирование. Масштабирование заключается в умножении преобразуемого числа на 10^{-N}, где N — количество десятичных знаков после запятой. Рассмотрим порядок действий.

1. Преобразовать число из ASCII-кода в упакованный десятичный формат как целое. Запятая при этом игнорируется, однако подсчитывается количество знаков после запятой N.
2. Число в упакованном формате занести в стек сопроцессора.
3. Умножить число на константу 10^{-N}. Для хранения констант в оперативной памяти можно использовать специальный массив вещественных чисел, индексом в котором служит N.
4. Результат извлечь из стека сопроцессора и сохранить в памяти в заданном вещественном формате.

При переводе в двоичный код десятичного числа, записанного в научном формате с порядком, равным K, умножение выполняется на 10^{K-N}, где N — количество знаков мантиссы числа после запятой.

При переводе вещественного числа из двоичного кода (X) в десятичный (D) также приходится выполнять дополнительные преобразования. После выполнения перевода число должно состоять из нормализованной мантиссы (M) и порядка (P):

$$D=M\times10^{P}$$

Порядок выполнения операций описан ниже.

1. Вычислить десятичный логарифм числа X. Целая часть результата представляет собой порядок числа P.

2. Разделить число X на 10^P. Результат является нормализованной мантиссой числа M.
3. Извлечь из стека математического сопроцессора M и P и сохранить в памяти в упакованном формате.
4. Преобразовать M и P из упакованного формата в ASCII-код.

Вычисление логарифмов осуществляется сопроцессором с помощью универсальной команды FYL2X (выполняющей вычисление значения $\log_2 X$ и умножение результата на Y). Десятичный логарифм вычисляется по формуле:

$$\log_{10}X=\log_{10}2\times\log_2 X$$

Для обеспечения максимальной точности преобразования в наборе команд сопроцессора есть группа операций загрузки в стек констант, наиболее часто используемых при вычислениях. Значение $\log_{10}2$ загружается командой FLDLG2.

Значение 10^P можно вычислить многими различными способами, в том числе — с помощью сопроцессора, однако в приведенных ниже примерах используется наиболее простой способ — табличный.

Все математические функции сгруппированы в одном модуле — в листинге 2.5. Назначение подпрограмм, включенных в листинг 2.5, следующее:

- процедура Int32_to_String осуществляет перевод 32-разрядного двоичного целого числа (со знаком) в текстовую строку, представляющую собой десятичное число в ASCII-коде;
- процедура String_to_Int32 осуществляет перевод целого десятичного числа, записанного в виде текстовой строки в ASCII-коде, в двоичное 32-разрядное число;
- процедуры ShowDecByte, ShowDecWord и ShowDecDWord осуществляют преобразование в десятичный код (при помощи функции Int32_to_String) и вывод в заданную позицию экрана 8-, 16- и 32-разрядных двоичных чисел соответственно;
- процедура DoubleFloat_to_String преобразует вещественное число из двоичного формата (с удвоенной точностью) в текстовую строку, представляющую собой десятичное число в формате с плавающей точкой (в ASCII-коде);
- процедура DoubleFloat_to_ExpForm преобразует вещественное число из двоичного формата (с удвоенной точностью) в текстовую строку, представляющую собой десятичное число в научном формате (то есть с мантиссой и экспонентой);

- процедура String_to_DoubleFloat осуществляет перевод вещественного десятичного числа, записанного в формате с плавающей точкой в виде текстовой строки в ASCII-коде, в двоичное число удвоенной точности;
- процедура BCD_to_ASCII преобразует целое десятичное число из кода BCD в ASCII-код (процедура является вспомогательной и не должна вызываться извне, то есть из программ, не принадлежащих к данному модулю);
- процедура ShowDataString предназначена для вывода в заданную позицию экрана текстовой строки, полученной в результате преобразования двоичного числа (любого типа) в десятичное;
- процедура GetInteger обеспечивает ввод с клавиатуры десятичного целого числа со знаком;
- процедура GetFloat обеспечивает ввод вещественного десятичного числа в формате с плавающей точой.

Листинг 2.5. Набор подпрограмм, предназначенных для перевода целых и вещественных чисел из двоичной системы счисления в десятичную и наоборот

```
DATASEG
; Количество разрядов мантиссы (1-18)
MaxPositions DW 10
; Количество знаков числа после запятой (1-17)
NumberSymbolsAD DW 5
; Машинный ноль
Data_Minimum DQ 1.E-110
; Константы (10 в степени N)
MConst DQ 1.0E1,1.0E2,1.0E3,1.0E4,1.0E5
       DQ 1.0E6,1.0E7,1.0E8,1.0E9,1.0E10
       DQ 1.0E11,1.0E12,1.0E13,1.0E14,1.0E15
       DQ 1.0E16,1.0E17,1.0E18,1.0E19,1.0E20
       DQ 1.0E21,1.0E22,1.0E23,1.0E24,1.0E25
       DQ 1.0E26,1.0E27,1.0E28,1.0E29,1.0E30
       DQ 1.0E31,1.0E32,1.0E33,1.0E34,1.0E35
       DQ 1.0E36,1.0E37,1.0E38,1.0E39,1.0E40
       DQ 1.0E41,1.0E42,1.0E43,1.0E44,1.0E45
       DQ 1.0E46,1.0E47,1.0E48,1.0E49,1.0E50
       DQ 1.0E51,1.0E52,1.0E53,1.0E54,1.0E55
       DQ 1.0E56,1.0E57,1.0E58,1.0E59,1.0E60
       DQ 1.0E61,1.0E62,1.0E63,1.0E64,1.0E65
       DQ 1.0E66,1.0E67,1.0E68,1.0E69,1.0E70
       DQ 1.0E71,1.0E72,1.0E73,1.0E74,1.0E75
       DQ 1.0E76,1.0E77,1.0E78,1.0E79,1.0E80
       DQ 1.0E81,1.0E82,1.0E83,1.0E84,1.0E85
```

продолжение ⇗

Листинг 2.5 (*продолжение*)

```
        DQ 1.0E86,1.0E87,1.0E88,1.0E89,1.0E90
        DQ 1.0E91,1.0E92,1.0E93,1.0E94,1.0E95
        DQ 1.0E96,1.0E97,1.0E98,1.0E99,1.0E100
        DQ 1.0E101,1.0E102,1.0E103,1.0E104,1.0E105
        DQ 1.0E106,1.0E107,1.0E108,1.0E109,1.0E110
        DQ 1.0E111,1.0E112,1.0E113,1.0E114,1.0E115
        DQ 1.0E116,1.0E117,1.0E118,1.0E119,1.0E120
        DQ 1.0E121,1.0E122,1.0E123,1.0E124,1.0E125
        DQ 1.0E126,1.0E127,1.0E128
; Данные передаются в математические процедуры
; через общую область памяти (главный сегмент данных)
; 32-разрядное целое число
Data_Int32    DD ?
; Число с плавающей точкой двойной точности
Data_Double   DQ ?
; Модуль числа с плавающей точкой
Data_Abs      DQ ?
; Число в BCD-формате
Data_BCD      DT ?
; Управляющее слово сопроцессора
Data_Control  DW ?
; Вспомогательный флаг
Data_Flag     DB ?
; Целая часть десятичного логарифма числа
Data_Log10    DW ?
; Знак результата (если не 0 - отрицательное число)
Data_Sign     DB ?
; Строка для хранения числа в коде ASCII
Data_String   DB 32 DUP (?)
; Структура для вывода результата
OutD_String   DB 34 DUP (?)
EVEN

CODESEG
;****************************************************
;* ПРЕОБРАЗОВАТЬ 32-РАЗРЯДНОЕ ЦЕЛОЕ ЧИСЛО В СТРОКУ *
;* Входные параметры:                               *
;* Data_Int32 - 32-разрядное число.                 *
;* Выходные параметры:                              *
;* Data_String - строка-результат.                  *
;****************************************************
PROC Int32_to_String near
        pushad
        push    DS
        push    ES
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
        mov     ES,AX
        cld
```

```
; Перевести число из двоичного кода в код BCD
        fninit              ;сброс сопроцессора
        ; Загрузить число в двоичном коде
        fild    [Data_Int32]
        ; Извлечь число в коде BCD
        fbstp   [Data_BCD]
; Результат записывать в строку Data_String
        mov     DI,offset Data_String
        call    BCD_to_ASCII
; Записать признак конца строки (код 0)
        xor     AL,AL
        stosb
        pop     ES
        pop     DS
        popad
        ret
ENDP Int32_to_String

;****************************************************
;*  ПРЕОБРАЗОВАТЬ СТРОКУ В 32-РАЗРЯДНОЕ ЦЕЛОЕ ЧИСЛО  *
;* Входные параметры:                               *
;* Data_String - число в коде ASCII.                *
;* Выходные параметры:                              *
;* Data_Int32 - 32-разрядное число в двоичном коде. *
;****************************************************
PROC String_to_Int32 near
        pushad
        push    DS
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
        cld
        ; Очищаем Data_BCD
        mov     [dword ptr Data_BCD],0
        mov     [dword ptr Data_BCD+4],0
        mov     [word ptr Data_BCD+8],0
        ; Очищаем байт знака
        mov     [Data_Sign],0
        ; Заносим в SI указатель на строку
        mov     SI,offset Data_String

        ; Пропускаем пробелы перед числом
        mov     CX,64 ;защита от зацикливания
@@ShiftIgnore:
        lodsb
        cmp     AL,' '
        jne     @@ShiftIgnoreEnd
        loop    @@ShiftIgnore
        jmp short @@Error
@@ShiftIgnoreEnd:
```

продолжение ⇒

Листинг 2.5 (*продолжение*)

```
        ; Проверяем знак числа
        cmp     AL,'-'
        jne     @@Positive
        mov     [Data_Sign],80h
        lodsb
@@Positive:
        ; 32-битное двоичное число соответствует
        ; десятичному, имеющему до 9 разрядов
        mov     CX,9
@@ASCIItoBCDConversion:
        ; Символы числа должны быть цифрами
        cmp     AL,'0'
        jb      @@Error
        cmp     AL,'9'
        ja      @@Error
        ; Пишем очередную цифру в младшую тетраду BCD
        and     AL,0Fh
        or      [byte ptr Data_BCD],AL
        ; Проверка на конец строки
        cmp     [byte ptr SI],0
        je      @@ASCIItoBCDConversionEnd
        ; Сдвигаем BCD на 4 разряда влево
        ; (сдвигаем старшие 2 байта)
        mov     AX,[word ptr Data_BCD+6]
        shld    [word ptr Data_BCD+8],AX,4
        ; (сдвигаем средние 4 байта)
        mov     EAX,[dword ptr Data_BCD]
        shld    [dword ptr Data_BCD+4],EAX,4
        ; (сдвигаем младшие 4 байта)
        shl     [dword ptr Data_BCD],4
        ; Загружаем следующий символ в AL
        lodsb
        loop    @@ASCIItoBCDConversion
        cmp     AL,0
        jne     @@Error ;переполнение разрядной сетки

; ПРЕОБРАЗОВАТЬ ЧИСЛО ИЗ КОДА BCD В ЦЕЛОЕ ЧИСЛО
@@ASCIItoBCDConversionEnd:
        ; Вписать знак в старший байт
        mov     AL,[Data_Sign]
        mov     [byte ptr Data_BCD+9],AL
        ; Сбросить регистры сопроцессора
        fninit
        ; Загрузить в сопроцессор число в BCD-формате
        fbld    [Data_BCD]
        ; Выгрузить число в двоичном формате
        fistp   [Data_Int32]
        jmp short @@End

@@Error:; При любой ошибке обнулить результат
```

```
        mov     [Data_Int32],0
@@End:  pop     DS
        popad
        ret
ENDP String_to_Int32

;*****************************************************
;*      ВЫВОД БАЙТА НА ЭКРАН В ДЕСЯТИЧНОМ КОДЕ       *
;* Подпрограмма выводит содержимое регистра AL в     *
;* шестнадцатеричном коде в указанную позицию экрана. *
;* Координаты позиции передаются через глобальные    *
;* переменные ScreenString и ScreenColumn. После     *
;* выполнения операции вывода байта происходит       *
;* автоматическое приращение значений этих переменных. *
;*****************************************************
PROC ShowDecByte NEAR
        push    EAX
        and     EAX,0FFh
        call    ShowDecDWord
        pop     EAX
        ret
ENDP ShowDecByte

;***************************************************
;*       ВЫВОД 16-РАЗРЯДНОГО СЛОВА НА ЭКРАН        *
;*               В ДЕСЯТИЧНОМ КОДЕ                 *
;* Параметры:                                      *
;* AX - число, которое будет выведено на экран.    *
;* Номер строки передается через глобальную        *
;* переменную ScreenString, номер столбца - через  *
;* переменную ScreenColumn, цвет текста определяется *
;* глобальной переменой TextColorAndBackground.    *
;***************************************************
PROC ShowDecWord NEAR
        push    EAX
        and     EAX,0FFFFh
        call    ShowDecDWord
        pop     EAX
        ret
ENDP ShowDecWord

;***************************************************
;*       ВЫВОД 32-РАЗРЯДНОГО СЛОВА НА ЭКРАН        *
;*               В ДЕСЯТИЧНОМ КОДЕ                 *
;* Параметры:                                      *
;* EAX - число, которое будет выведено на экран.   *
;* Номер строки передается через глобальную        *
;* переменную ScreenString, номер столбца - через  *
;* переменную ScreenColumn, цвет текста определяется *
;* глобальной переменой TextColorAndBackground.    *
;***************************************************
```

продолжение ⇗

Листинг 2.5 (*продолжение*)

```
PROC ShowDecDWord NEAR
        pushad
        push    DS
        push    ES
; Настроить регистр DS на глобальный сегмент данных
        mov     DI,[CS:MainDataSeg]
        mov     DS,DI
; Перевести число в десятичный код
        mov     [Data_Int32],EAX
        call    Int32_to_String
; Настроить регистры ES:DI для
; прямого вывода в видеопамять
        ; Загрузить адрес сегмента видеоданных в ES
        mov     AX,0B800h
        mov     ES,AX
        ; Умножить номер строки на длину
        ; строки в байтах
        mov     AX,[ScreenString]
        mov     DX,160
        mul     DX
        ; Прибавить к полученному произведению номер
        ; колонки (дважды)
        add     AX,[ScreenColumn]
        add     AX,[ScreenColumn]
        ; Переписать результат в индексный регистр
        mov     DI,AX
        cld
; Использовать цвет символов, заданный по умолчанию
        mov     AH,[TextColorAndBackground]
; Вывести число на экран
        mov     CX,10
        mov     SI,offset Data_String
@@NextChar:
        lodsb               ;загрузить цифру в AL
        and     AL,AL       ;проверка на 0 (конец строки)
        jz      @@EndOfString
        stosw               ;вывести цифру на экран
        loop    @@NextChar
@@EndOfString:
        pop     ES
        pop     DS
        popad
        ret
ENDP ShowDecDWord

;*****************************************************
;*  ПРЕОБРАЗОВАНИЕ ЧИСЛА С ПЛАВАЮЩЕЙ ТОЧКОЙ В СТРОКУ   *
;* Число имеет формат с удвоенной точностью, результат *
;* выдается в десятичном коде, в "бытовом" формате с   *
```

```
;* фиксированным количеством знаков после запятой.    *
;* Входные параметры:                                  *
;* Data_Double - преобразуемое число;                  *
;* NumberSymbolsAD - количество знаков после           *
;*                   запятой (0-17).                   *
;* Выходные параметры:                                 *
;* Data_String - строка-результат.                     *
;******************************************************
PROC DoubleFloat_to_String near
        pushad
        push    DS
        push    ES
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
        mov     ES,AX
        ; Результат записывать в строку Data_String
        mov     DI,offset Data_String

        ; Сдвигаем число влево на NumberSymbolsAD
        ; десятичных разрядов
        fninit                  ;сброс сопроцессора
        fld     [Data_Double]   ;загрузить число
        mov     BX,[NumberSymbolsAD]
        cmp     BX,0
        je      @@NoShifts      ;нет цифр после запятой
        jl      @@Error         ;ошибка
        dec     BX
        shl     BX,3            ;умножаем на 8
        add     BX,offset MConst
        fmul    [qword ptr BX]  ;умножить на константу
@@NoShifts:
        ; Извлечь число в коде BCD
        fbstp   [Data_BCD]
; Проверить результат на переполнение
        mov     AX,[offset Data_BCD + 8]
        cmp     AX,0FFFFh  ;"десятичное" переполнение?
        je      @@Overflow
; Выделить знак числа и записать его в ASCII-коде
        mov     AL,[offset Data_BCD + 9]
        and     AL,AL
        jz      @@NoSign
        mov     AL,'-'
        stosb
@@NoSign:
; Распаковать число в код ASCII
        mov     BX,8      ;смещение последней пары цифр
        mov     CX,9      ;счетчик пар цифр
        ; Определить позицию десятичной точки в числе
        mov     DX,18
        sub     DX,[NumberSymbolsAD]
```

продолжение

Листинг 2.5 (*продолжение*)

```
        js      @@Error  ;ошибка, если отрицательная
        jz      @@Error  ;или нулевая позиция
@@NextPair:
        ; Загрузить очередную пару разрядов
        mov     AL,[BX + offset Data_BCD]
        mov     AH,AL
        ; Выделить, перевести в ASCII и
        ; сохранить старшую тетраду
        shr     AL,4
        add     AL,'0'
        stosb
        dec     DX
        jnz     @@N0
        mov     AL,'.'
        stosb
@@N0:   ; Выделить, перевести в ASCII и
        ; сохранить младшую тетраду
        mov     AL,AH
        and     AL,0Fh
        add     AL,'0'
        stosb
        dec     DX
        jnz     @@N1
        mov     AL,'.'
        stosb
@@N1:   dec     BX
        loop    @@NextPair
        mov     AL,0
        stosb

; Убрать незначащие нули слева
        mov     DI,offset Data_String
        mov     SI,offset Data_String
        ; Пропустить знак числа, если он есть
        cmp     [byte ptr SI],'-'
        jne     @@N2
        inc     SI
        inc     DI
@@N2:   ; Загрузить в счетчик цикла количество разрядов
        ; числа плюс 1 (байт десятичной точки)
        mov     CX,18+1+1
        ; Пропустить незначащие нули
@@N3:   cmp     [byte ptr SI],'0'
        jne     @@N4
        cmp     [byte ptr SI+1],'.'
        je      @@N4
        inc     SI
        loop    @@N3
```

```
        ; Ошибка - нет значащих цифр
        jmp short @@Error
; Скопировать значащую часть числа в начало строки
@@N4:   rep movsb
        jmp short @@End

; Ошибка
@@Error:
        mov     AL,'E'
        stosb
        mov     AL,'R'
        stosb
        mov     AL,'R'
        stosb
        xor     AL,AL
        stosb
        jmp short @@End
; Переполнение разрядной сетки
@@Overflow:
        mov     AL,'#'
        stosb
        xor     AL,AL
        stosb
; Конец процедуры
@@End:  pop     ES
        pop     DS
        popad
        ret
ENDP DoubleFloat_to_String

;*****************************************************
;*  ПРЕОБРАЗОВАТЬ ЧИСЛО С ПЛАВАЮЩЕЙ ТОЧКОЙ В СТРОКУ    *
;* Число имеет формат с удвоенной точностью, результат *
;* выдается в десятичном коде в научном формате, т.е.  *
;* с нормализованной мантиссой и порядком.             *
;* Входные параметры:                                  *
;* Data_Double - преобразуемое число;                  *
;* MaxPositions - количество разрядов мантиссы (1-18). *
;* Выходные параметры:                                 *
;* Data_String - строка-результат.                     *
;*****************************************************
PROC DoubleFloat_to_ExpForm near
        pushad
        push    DS
        push    ES
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
        mov     ES,AX
```

продолжение

Листинг 2.5 (*продолжение*)

```
        ; Результат записывать в строку Data_String
        mov     DI,offset Data_String
; Определить знак числа
        fninit                  ;сброс сопроцессора
        fld     [Data_Double]   ;загрузить число
        fxam                    ;протестировать число
        fstsw   AX
        test    ah,10b          ;проверить знак
        jz      @@z0
        ; Число отрицательное
        mov     AL,'-'          ;записать знак "минус"
        stosb
; Запомнить модуль числа
        fabs
@@z0:   fst     [Data_Abs]
; Произвести проверку на "машинный ноль"
        fcom    [Data_Minimum]
        fstsw   AX
        and     AH,1000001b
        jnz     @@Zero

; Устанавить режим округления "вниз" (01)
        fstcw   [Data_Control] ;считать слово состояния
        and     [Data_Control],0F7FFh
        or      [Data_Control],0400h
        fldcw   [Data_Control]

; Вычислить десятичный логарифм числа
        fldlg2
        fxch    ST(1)
        fyl2x
        ; Записать десятичный порядок
        fistp   [Data_Log10]

; Устанавить режим округления "к ближайшему" (00)
        fstcw   [Data_Control] ;считать слово состояния
        and     [Data_Control],0F3FFh
        fldcw   [Data_Control]

; Нормализовать десятичную мантиссу
        fninit                  ;сброс сопроцессора
        fld     [Data_Abs]      ;загрузить модуль числа
        mov     BX,[MaxPositions]
        sub     BX,[Data_Log10]
        cmp     BX,0
        je      @@z2            ;сдвиги не нужны
        jl      @@z1            ;нужен сдвиг вправо
        ; Сдвинуть число влево
        ; на [BX] десятичных разрядов
```

```
        dec     BX
        cmp     BX,127
        ja      @@Zero          ;машинный ноль
        shl     BX,3            ;умножаем на 8
        add     BX,offset MConst
        fmul    [qword ptr BX] ;умножить на константу
        jmp short @@z2
        ; Сдвинуть число вправо
        ; на [BX] десятичных разрядов
@@z1:   neg     BX
        dec     BX
        cmp     BX,127
        ja      @@Overflow      ;переполнение
        shl     BX,3            ;умножаем на 8
        add     BX,offset MConst
        fdiv    [qword ptr BX] ;разделить на константу
@@z2:   fbstp   [Data_BCD]      ;извлечь число в коде BCD

; Записать точку перед мантиссой
        mov     SI,DI  ;запомнить позицию точки
        mov     AL,'.'
        stosb
; Преобразовать мантиссу в код ASCII
        call    BCD_to_ASCII
; Поменять местами точку и первую цифру мантиссы
        mov     AX,[SI]
        xchg    AL,AH
        mov     [SI],AX

; Перевести порядок результата в код BCD
        ; Проверить значение порядка
        cmp     [Data_Log10],0
        je      @@End
        ; Записать признак порядка
        mov     AL,'e'
        stosb
        fninit                  ;сброс сопроцессора
        ; Загрузить десятичный порядок в двоичном коде
        fild    [Data_Log10]
        ; Извлечь порядок в коде BCD
        fbstp   [Data_BCD]
        ; Результат записать в строку Data_String
        call    BCD_to_ASCII
        jmp short @@End

@@Zero: ; Машинный ноль
        mov     AL,'0'
        stosb
        jmp short @@End
@@Overflow:
```

продолжение ⇗

Листинг 2.5 (*продолжение*)

```
        ; Переполнение разрядной сетки
        mov     AL,'#'
        stosb
@@End:  ; Записать признак конца строки (код 0)
        xor     AL,AL
        stosb

        pop     ES
        pop     DS
        popad
        ret
ENDP DoubleFloat_to_ExpForm

;****************************************************
;* ПРЕОБРАЗОВАТЬ СТРОКУ В ЧИСЛО С ПЛАВАЮЩЕЙ ТОЧКОЙ  *
;*      (число имеет обычный, "бытовой" формат)     *
;* Входные параметры:                               *
;* Data_String - число в коде ASCII.                *
;* Выходные параметры:                              *
;* Data_Double - число в двоичном коде.             *
;****************************************************
PROC String_to_DoubleFloat near
        pushad
        push    DS
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
        cld
        ; Очищаем Data_BCD
        mov     [dword ptr Data_BCD],0
        mov     [dword ptr Data_BCD+4],0
        mov     [word ptr Data_BCD+8],0
        ; Очищаем байт знака
        mov     [Data_Sign],0
        ; Заносим в SI указатель на строку
        mov     SI,offset Data_String
        ; Пропускаем пробелы перед числом
        mov     CX,64 ;защита от зацикливания
@@ShiftIgnore:
        lodsb
        cmp     AL,' '
        jne     @@ShiftIgnoreEnd
        loop    @@ShiftIgnore
        jmp     @@Error
@@ShiftIgnoreEnd:
        ; Проверяем знак числа
        cmp     AL,'-'
        jne     @@Positive
        mov     [Data_Sign],80h
        lodsb
```

```
@@Positive:
        mov     [Data_Flag],0 ;признак наличия точки
        mov     DX,0          ;позиция точки
        mov     CX,18         ;макс. число разрядов
@@ASCIItoBCDConversion:
        cmp     AL,'.'        ;точка?
        jne     @@NotDot
        cmp     [Data_Flag],0 ;точка не встречалась?
        jne     @@Error
        mov     [Data_Flag],1
        lodsb
        cmp     AL,0          ;конец строки?
        jne     @@NotDot
        jmp     @@ASCIItoBCDConversionEnd
@@NotDot:
        ; Увеличить на 1 значение позиции точки,
        ; если она еще не встречалась
        cmp     [Data_Flag],0
        jnz     @@Figures
        inc     DX
@@Figures:
        ; Символы числа должны быть цифрами
        cmp     AL,'0'
        jb      @@Error
        cmp     AL,'9'
        ja      @@Error
        ; Пишем очередную цифру в младшую тетраду BCD
        and     AL,0Fh
        or      [byte ptr Data_BCD],AL
        ; Проверка на конец строки
        cmp     [byte ptr SI],0
        je      @@ASCIItoBCDConversionEnd
        ; Сдвигаем BCD на 4 разряда влево
        ; (сдвигаем старшие 2 байта)
        mov     AX,[word ptr Data_BCD+6]
        shld    [word ptr Data_BCD+8],AX,4
        ; (сдвигаем средние 4 байта)
        mov     EAX,[dword ptr Data_BCD]
        shld    [dword ptr Data_BCD+4],EAX,4
        ; (сдвигаем младшие 4 байта)
        shl     [dword ptr Data_BCD],4
        ; Загружаем следующий символ в AL
        lodsb
        loop    @@ASCIItoBCDConversion
        ; Если 19-й символ не 0 и не точка,
        ; то ошибка переполнения
        cmp     AL,'.'
        jne     @@NotDot2
        inc     CX
        lodsb
```

продолжение

Листинг 2.5 (*продолжение*)

```
@@NotDot2:
        cmp     AL,0
        jne     @@Error ;переполнение разрядной сетки

; ПРЕОБРАЗОВАТЬ ЧИСЛО ИЗ КОДА BCD В ВЕЩЕСТВЕННОЕ ЧИСЛО
@@ASCIItoBCDConversionEnd:
        ; Вписать знак в старший байт
        mov     AL,[Data_Sign]
        mov     [byte ptr Data_BCD+9],AL
        ; Сбросить регистры сопроцессора
        fninit
        ; Загрузить в сопроцессор число в BCD-формате
        fbld    [Data_BCD]
        ; Вычислить номер делителя
        mov     BX,18+1
        sub     BX,CX
        sub     BX,DX
        cmp     BX,0
        je      @@NoDiv
        dec     BX
        shl     BX,3            ;умножаем на 8
        add     BX,offset MConst
        fdiv    [qword ptr BX] ;разделить на константу
@@NoDiv:; Выгрузить число в двоичном формате
        fstp    [Data_Double]
        jmp short @@End

@@Error:; При любой ошибке обнулить результат
        fldz    ;занести ноль в стек сопроцессора
        fstp    [Data_Double]
@@End:  pop     DS
        popad
        ret
ENDP String_to_DoubleFloat

;*****************************************************
;*   ПРЕОБРАЗОВАТЬ ЧИСЛО ИЗ КОДА BCD В КОД ASCII     *
;* (вспомогательная функция, регистры не сохраняет) *
;* Входные параметры:                                *
;* Data_BCD - число в BCD-формате.                   *
;* Регистр DI - указатель на строку результата.     *
;*****************************************************
PROC BCD_to_ASCII near
; Выделить знак числа и записать его в ASCII-коде
        mov     AL,[offset Data_BCD + 9]
        and     AL,AL
        jz      @@n0
        mov     AL,'-'
        stosb
```

```
; Пропустить незначащие (нулевые) разряды слева
@@n0:   mov     BX,B
        mov     CX,9
@@n1:   ;проверяем на 0 очередную пару разрядов
        cmp     [byte ptr BX+offset Data_BCD],0
        jne     @@n2
        dec     BX
        loop    @@n1
; Если значение числа равно нулю, записать символ
; нуля в строку результата и выйти из программы
        mov     AL,'0'
        stosb
        jmp short @@End
; Пропустить незначащий ноль в старшей
; тетраде (если он есть)
@@n2:   ; Загрузить первую значащую пару разрядов
        mov     AL,[BX + offset Data_BCD]
        mov     AH,AL
        ; Выделить, перевести в ASCII и
        ; сохранить старшую тетраду
        shr     AL,4
        cmp     AL,0
        ; Если 0 - пропустить старшую тетраду
        je      @@n3
        add     AL,'0'
        stosb
        ; Выделить, перевести в ASCII и
        ; сохранить младшую тетраду
@@n3:   mov     AL,AH
        and     AL,0Fh
        add     AL,'0'
        stosb
        dec     BX
        dec     CX
        jz      @@End ;выход, если это последний разряд

; Распаковать остальные разряды числа (если они есть)
@@n4:   ; Загрузить очередную пару разрядов
        mov     AL,[BX + offset Data_BCD]
        mov     AH,AL
        ; Выделить, перевести в ASCII и
        ; сохранить старшую тетраду
        shr     AL,4
        add     AL,'0'
        stosb
        ; Выделить, перевести в ASCII и
        ; сохранить младшую тетраду
        mov     AL,AH
        and     AL,0Fh
```

продолжение ⇗

Листинг 2.5 (*продолжение*)

```
        add     AL,'0'
        stosb
        dec     BX
        loop    @@n4
@@End:
        ret
ENDP BCD_to_ASCII

;***********************************************
;*      ОТОБРАЗИТЬ ЧИСЛО В КОДЕ ASCII НА ЭКРАН     *
;* Подпрограмма отображает Data_String на экран.  *
;* Координаты позиции передаются через глобальные *
;* переменные ScreenString и ScreenColumn. После  *
;* выполнения операции происходит автоматическое   *
;* приращение значений этих переменных.          *
;***********************************************
PROC ShowDataString near
        pusha
        push    ES
        mov     AX,[CS:MainDataSeg]
        mov     ES,AX
        cld
; Занести координаты сторки
        mov     DI,offset OutD_String
        mov     AX,[ScreenString]
        stosb
        mov     AX,[ScreenColumn]
        stosb
; Копировать Data_String в OutD_String
        mov     SI,offset Data_String
        mov     CX,31
        rep     movsb
; Поставить символ-ограничитель
        mov     AL,0
        stosb
        mov     SI,offset OutD_String
        call    ShowString
        pop     ES
        popa
        ret
ENDP ShowDataString

;************************************************
;*          ВВЕСТИ ЦЕЛОЕ ЧИСЛО С КЛАВИАТУРЫ        *
;* Подпрограмма обеспечивает ввод 9-разрядного     *
;* десятичного целого числа со знаком.            *
;* Координаты поля ввода задаются через переменные *
;* ScreenString и ScreenColumn. Цвет определяется  *
;* переменной TextColorAndBackground.              *
;************************************************
```

```
PROC GetInteger near
        pushad
; Инициализировать переменные
@@ClearField:
        ; Обнулить счетчик цифр
        mov     BX,0
        ; Установить координаты поля ввода
        mov     DI,offset OutD_String
        mov     AX,[ScreenString]
        mov     [DI],AL
        inc     DI
        mov     AX,[ScreenColumn]
        mov     [DI],AL
        inc     DI
        ; Очистить строку (заполнить пробелами)
        mov     [dword ptr DI],20202020h
        mov     [dword ptr DI+4],20202020h
        mov     [word ptr DI+8],2020h
        mov     [byte ptr DI+10],0 ;конец строки

; Цикл ввода
@@GetNextChar:
        ; Отобразить число на экране
        mov     SI,offset OutD_String
        call    ShowString
        ; Установить курсор в позицию ввода
        push    [ScreenColumn]
        add     [ScreenColumn],BX
        call    SetCursorPosition
        pop     [ScreenColumn]
        ; Принять байт с клавиатуры
        call    GetChar
        ; Цифра или команда?
        cmp     AL,0
        jz      @@Command  ;если ноль - введена команда
        ; Проверка на специальные символы
        cmp     AL,'-'
        je      @@M
        ; Проверка на диапазон '0'-'9'
        cmp     AL,'0'
        jb      @@Error
        cmp     AL,'9'
        ja      @@Error
        ; Проверяем количество символов (не более 10)
@@M:    cmp     BX,10
        jae     @@Error
        inc     BX
        ; Записываем символ в число
        mov     [DS:DI],AL
        inc     DI
        jmp short @@GetNextChar
```

продолжение ⇨

Листинг 2.5 (*продолжение*)

```
; Проанализировать код команды
@@Command:
        cmp     AH,B_Esc        ;очистить поле
        je      @@ClearField
@@TestRubout:
        cmp     AH,B_RUBOUT     ;"Забой"
        jne     @@TestEnter
        cmp     BX,0
        je      @@Error
        ; Передвинуть признак конца строки
        ; на разряд влево
        dec     DI
        dec     BX
        mov     [byte ptr DS:DI],' '
        jmp short @@GetNextChar
@@TestEnter:
        cmp     AH,B_Enter      ;завершение ввода числа
        jne     @@Error
        ; Перевести число в двоичный код
        mov     SI,offset Data_String
        mov     DI,offset OutD_String
        add     DI,2
        mov     CX,10
@@i1:   mov     AL,[DI]
        cmp     AL,' '          ;конец числа?
        jbe     @@EndOfString
        mov     [SI],AL
        inc     SI
        inc     DI
        loop    @@i1
@@EndOfString:
        mov     [byte ptr SI],0 ;конец строки
        call    String_to_Int32
        jmp short @@End

; Ошибка при вводе
@@Error:
        call    Beep
        jmp     @@GetNextChar

@@End:  popad
        ret
ENDP GetInteger

;***************************************************
;*      ВВЕСТИ ВЕЩЕСТВЕННОЕ ЧИСЛО С КЛАВИАТУРЫ     *
;* Подпрограмма обеспечивает ввод вещественного    *
```

```
;* числа в обычном формате (до 18 цифр).          *
;* Координаты поля ввода задаются через переменные *
;* ScreenString и ScreenColumn. Цвет определяется  *
;* переменной TextColorAndBackground.              *
;**************************************************
PROC GetFloat near
        pushad
; Инициализировать переменные
@@ClearField:
        ; Обнулить счетчик цифр
        mov     BX,0
        ; Установить координаты поля ввода
        mov     DI,offset OutD_String
        mov     AX,[ScreenString]
        mov     [DI],AL
        inc     DI
        mov     AX,[ScreenColumn]
        mov     [DI],AL
        inc     DI
        ; Очистить строку (заполнить пробелами)
        mov     [dword ptr DI],20202020h
        mov     [dword ptr DI+4],20202020h
        mov     [dword ptr DI+8],20202020h
        mov     [dword ptr DI+12],20202020h
        mov     [dword ptr DI+16],20202020h
        mov     [byte ptr DI+20],0 ;конец строки
; Цикл ввода
@@GetNextChar:
        ; Отобразить число на экране
        mov     SI,offset OutD_String
        call    ShowString
        ; Установить курсор в позицию ввода
        push    [ScreenColumn]
        add     [ScreenColumn],BX
        call    SetCursorPosition
        pop     [ScreenColumn]
        ; Принять байт с клавиатуры
        call    GetChar
        ; Цифра или команда?
        cmp     AL,0
        jz      @@Command  ;если ноль - введена команда
        ; Проверка на специальные символы
        cmp     AL,'-'
        je      @@M
        cmp     AL,'.'
        je      @@M
        ; Проверка на диапазон '0'-'9'
        cmp     AL,'0'
```

продолжение ⇨

Листинг 2.5 (*продолжение*)

```
        jb      @@Error
        cmp     AL,'9'
        ja      @@Error
        ; Проверяем количество символов (не более 20)
@@M:    cmp     BX,20
        jae     @@Error
        inc     BX
        ; Записываем символ в число
        mov     [DS:DI],AL
        inc     DI
        jmp short @@GetNextChar
; Проанализировать код команды
@@Command:
        cmp     AH,B_Esc        ;очистить поле
        je      @@ClearField
@@TestRubout:
        cmp     AH,B_RUBOUT     ;"Забой"
        jne     @@TestEnter
        cmp     BX,0
        je      @@Error
        ; Передвинуть признак конца строки
        ; на разряд влево
        dec     DI
        dec     BX
        mov     [byte ptr DS:DI],' '
        jmp short @@GetNextChar
@@TestEnter:
        cmp     AH,B_Enter      ;завершение ввода числа
        jne     @@Error
        ; Перевести число в двоичный код
        mov     SI,offset Data_String
        mov     DI,offset DutD_String
        add     DI,2
        mov     CX,20
@@i1:   mov     AL,[DI]
        cmp     AL,' '          ;конец числа?
        jbe     @@EndOfString
        mov     [SI],AL
        inc     SI
        inc     DI
        loop    @@i1
@@EndOfString:
        mov     [byte ptr SI],0 ;конец строки
        call    String_to_DoubleFloat
        jmp short @@End

; Ошибка при вводе
```

```
@@Error:
        call    Beep
        jmp     @@GetNextChar

@@End:  popad
        ret
ENDP GetFloat
ENDS
```

ПРИМЕЧАНИЕ

Обратите внимание: математический сопроцессор автоматизирует перевод числа из упакованного двоично-десятичного кода BCD в двоичный и наоборот, но при этом программист все еще должен самостоятельно выполнять преобразование из ASCII-кода в код BCD, а также контролировать корректность поступающей информации. Вводимое число не должно содержать посторонних символов и его значение не должно выходить за пределы некоторого заданного диапазона; для выводимых чисел также необходимо выполнять проверку на диапазон.

Программа MathFunctionsTest, приведенная в листинге 2.6, предназначена для тестирования функций перевода целых и вещественных чисел. Тестирование выполняется по методу *обратной связи*: введенное оператором число переводится в двоичный код, а затем из двоичного кода переводится обратно в десятичный и отображается на экран.

Листинг 2.6. Тестирование подпрограмм перевода вещественных чисел

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
Txt1 DB LIGHTMAGENTA,0,18
     DB "Тестирование подпрограмм перевода чисел",0
     DB LIGHTGREEN,9,0,"Введите целое число:",0
```

продолжение

Листинг 2.6 (*продолжение*)

```
Txt2 DB LIGHTGREEN,10,0
     DB "Число в шестнадцатеричном коде:",0
     DB LIGHTGREEN,11,0
     DB "Обратный перевод в десятичный код:",0
     DB LIGHTGREEN,13,0
     DB "Введите вещественное число:",0
Txt3 DB LIGHTGREEN,14,0,"Число в научном формате:",0
     DB LIGHTGREEN,15,0,"Число в обычном формате:",0
     DB YELLOW,24,29,"Нажмите любую клавишу",0
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC MathFunctionsTest
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Вывести заголовок
        MShowColorText 2,Txt1
; Задать цвет символов числа
        mov     [TextColorAndBackground],LIGHTGREY

; Ввести и вывести целое число
        ; Ввести целое число
        mov     [ScreenString],9
        mov     [ScreenColumn],21
        call    GetInteger
        ; Вывести комментарии
        MShowColorText 3,Txt2
        ; Вывести число в шестнадцатеричном коде
        MShowHexDWord 10,32,[Data_Int32]
        ; Вывести число в десятичном коде
        mov     [ScreenString],11
        mov     [ScreenColumn],35
        call    Int32_to_String
        call    ShowDataString

; Ввести и вывести вещественное число
        ; Ввести целое число
        mov     [ScreenString],13
        mov     [ScreenColumn],28
        call    GetFloat
        ; Вывести комментарии
```

```
        MShowColorText 3,Txt3
        ; Вывести число в научном формате
        mov     [ScreenString],14
        mov     [ScreenColumn],25
        call    DoubleFloat_to_ExpForm
        call    ShowDataString
        ; Вывести число в обычном формате
        mov     [ScreenString],15
        mov     [ScreenColumn],25
        call    DoubleFloat_to_String
        call    ShowDataString

        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
        call    GetChar

; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP MathFunctionsTest
ENDS
; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуры перевода чисел
include "list2_05.inc"

END
```

Использование счетчика тактов в качестве таймера

Время — наиболее точно измеряемая в настоящее время физическая величина. Современный персональный компьютер содержит большое количество разнообразных таймеров, часть из которых доступна для обращений со стороны операционной системы и прикладных программ.

В литературе описана работа только с двумя самыми старыми типами таймеров: системным таймером и часами реального времени CMOS. Таймеры обоих типов не лишены определенных недостатков:

- часы CMOS не позволяют измерять короткие интервалы времени и не всегда корректно вырабатывают прерывание;

- системный таймер в обычном режиме работы не измеряет интервалы короче 1/18 секунды, а при его перепрограммировании в другой режим сбиваются системные часы и перестают нормально функционировать устройства, использующие информацию от системного таймера (например, накопители гибких дисков).

Между тем современные процессоры типа Intel x86 содержат в своем составе гораздо более простое, точное и эффективное средство измерения времени — *счетчик тактов* (Time Stamp Counter). Указанный счетчик введен в состав процессоров: начиная с изделий класса Pentium, он присутствует во всех последующих модификациях процессоров Intel (и скопированных с них изделиях-клонах производства других фирм). Счетчик тактов входит в состав блока мониторинга производительности процессора вместе с несколькими другими специализированными счетчиками. Однако на данный момент из всех регистров блока мониторинга только счетчик тактов можно использовать без риска возникновения несовместимости с последующими моделями процессоров (структура счетчика проста и потому стабильна).

Счетчик тактов является 64-разрядным. Минимальный и максимальный измеряемые интервалы времени зависят от внутренней тактовой частоты процессора. Максимальный измеряемый период времени выбран с очень большим запасом — интервал 2^{64} тактов даже при частоте 1 ГГц будет соответствовать периоду приблизительно в 585 лет.

Минимальный измеряемый интервал при внутренней частоте 100 МГц составляет 10 нс, а при частоте 1 ГГц — 1 нс. Поскольку этот интервал зависит от внутренней тактовой частоты процессора, необходимо провести калибровку таймера, построенного на базе счетчика тактов, по системным часам или часам CMOS. Системные часы (так же, как и часы CMOS) имеют свой собственный встроенный кварцевый генератор, они не зависят от частоты системной шины и внутренней частоты процессора, то есть выдают правильные показания даже в случае разгона системы.

Доступ к счетчику тактов контролируется флагом TSD в управляющем регистре CR4 процессора (если флаг сброшен, команда выполняется при любом уровне привилегий выполняемой программы, а если установлен — то только при уровне 0). Счетчик обнуляется в момент сброса процессора, после чего начинает подсчет тактов, прошедших с момента сброса. Прочитать содержимое счетчика так-

тов можно, используя команду RDTSC. После ее выполнения в регистре EDX будет размешена старшая часть, а в регистре EAX — младшая часть значения, которое счетчик тактов содержал в момент выполнения команды. Таким образом, реализация программного таймера существенно упрощается — можно сразу выполнить операцию деления на 32-разрядный делитель (однако в этом случае значение делителя должно быть достаточно велико, чтобы не происходило переполнения, то есть частное не должно содержать более 32 двоичных разрядов).

Программа ProcFrequency, приведенная в листинге 2.7, использует счетчик тактов для измерения реальной внутренней тактовой частоты процессора. Вначале измеряется длительность (в тактах процессора) шестнадцати временных интервалов между отсчетами системного таймера, после чего вычисляется средняя длительность интервала. Полученное усредненное значение умножается на частоту тактового генератора системного таймера (1 193 180 Гц), а затем делится на коэффициент пересчета системного таймера (65 536). Результат нужно разделить на константу 1 000 000, чтобы получить значение внутренней частоты процессора в мегагерцах. При выводе на экран полученного значения частоты используется процедура перевода целых чисел из двоичного кода в десятичный из листинга 2.5. Вспомогательная процедура WaitTimerStateChange выполняет цикл опроса ячейки памяти области данных BIOS, которую системный таймер использует в качестве счетчика тактов (процедура ожидает изменения значения данного счетчика).

Листинг 2.7. Использование счетчика тактов для определения внутренней тактовой частоты процессора

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS
```

продолжение ⇨

Листинг 2.7 (*продолжение*)

```
DATASEG
; Предыдущее значение системного таймера
Time        DD ?
; Значение счетчика тактов в момент времени t0
T0Count     DD ?
; Текстовые сообщения
Text DB LIGHTGREEN,0,20
     DB "Определение тактовой частоты процессора",0
     DB LIGHTGREEN,1,28,"при помощи команды RDTSC",0
     DB LIGHTMAGENTA,12,20
     DB "Тактовая частота процессора (МГц):",0
     DB LIGHTGREEN,24,35,"Ждите ...",0
AnyK DB YELLOW,24,29,"Нажмите любую клавишу",0
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC ProcFrequency
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 4,Text
; Использовать для вывода чисел белый цвет, черный фон
        mov     [TextColorAndBackground],WHITE
; Настроить сегментный регистр ES на область данных BIOS
        mov     AX,0
        mov     ES,AX
; Ожидание изменения состояния таймера
        mov     EAX,[ES:046Ch]
        mov     [Time],EAX
        call    WaitTimerStateChange
; Запомнить начальное значение счетчика тактов
        DB      0Fh,31h  ;команда RDTSC
        mov     [T0Count],EAX
; Ожидать прохождения шестнадцати интервалов
        mov     CX,16
@@ti:   call    WaitTimerStateChange
        loop    @@ti
; Запомнить начальное значение счетчика тактов
```

```
        DB      0Fh,31h  ;команда RDTSC
; Вычислить среднюю длительность интервала
        sub     EAX,[T0Count]
        shr     EAX,4    ;деление на 16
; Вычислить тактовую частоту процессора
        ; Умножить среднюю длительность интервала на
        ; частоту тактового генератора таймера
        mov     EDX,1193180
        mul     EDX
        ; Разделить результат на коэффициент
        ; пересчета системного таймера (65536)
        shrd    EAX,EDX,16
        ; Вычислить частоту в МГц (разделить на 1000000)
        xor     EDX,EDX
        mov     EBX,1000000
        div     EBX
        ; Вывести результат в десятичном коде
        mov     [Data_Int32],EAX
        mov     [ScreenString],12
        mov     [ScreenColumn],55
        call    Int32_to_String
        call    ShowDataString
; Ожидать нажатия любой клавиши
        MShowColorString AnyK
        call    GetChar
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP ProcFrequency
;*************************************************
;* ОЖИДАНИЕ ОЧЕРЕДНОГО ИЗМЕНЕНИЯ ЗНАЧЕНИЯ ТАЙМЕРА *
;*************************************************
PROC WaitTimerStateChange near
        mov     EAX,[Time]
@@T:    cmp     EAX,[ES:046Ch]
        je      @@T
        mov     EAX,[ES:046Ch]
        mov     [Time],EAX
        ret
ENDP WaitTimerStateChange
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуры перевода чисел
include "list2_05.inc"

END
```

ВНИМАНИЕ

При выполнении компоновки примеров из данного раздела необходимо разрешить использование привилегированных команд процессора i386, то есть команду TLINK нужно запускать с ключом /3.

Например, ассемблирование и компоновка листинга 2.3 выполняются при помощи следующих команд:

```
tasm lst_2_03.asm
tlink lst_2_03/3
```

В следующих главах все примеры, использующие режим линейной адресации памяти, также нужно компоновать с ключом /3.

ПРИМЕЧАНИЕ

Для запуска всех приведенных в данном разделе примеров, кроме последнего, достаточно иметь процессор уровня i486DX. Пример из листинга 2.7 может быть запущен только на процессоре класса Pentium, поскольку более ранние модификации не имели встроенного счетчика тактов.

Глава 3
Работа с устройствами, подключенными к шине PCI

Документацию по работе с шиной PCI распространяет организация PCI Special Interest Group (http://www.pcisig.com). Основные стандарты доступны по Сети только членам организации, а всем остальным они высылаются в виде книг или на компакт-дисках (услуга платная). В то же время документацию по шине PCI [82] совершенно бесплатно можно скачать с серверов, посвященных операционной системе Linux.

СОВЕТ

Сайты Интернет с компьютерной документацией очень часто (несколько раз в год) реорганизуются и переименовываются, поэтому поиск документов лучше вести по имени автора или названию с помощью поисковых систем. Так как компьютерная документация обычно представлена в формате Adobe Acrobat, удобнее всего использовать поисковые системы, позволяющие в запросе указывать расширение файла (в данном случае — .pdf).

Конфигурационное пространство устройства PCI

Для программистов интерес в первую очередь представляют функции PCI BIOS, поскольку они позволяют получить доступ к информации об адресном пространстве и пространстве ввода-вывода подключенных к шине PCI устройств. Описание этих функций дано в PCI BIOS Specification [81], а также в руководстве PhoenixBIOS User's Manual [85].

Интерфейс PCI BIOS обеспечивает аппаратно-независимый метод управления устройствами PCI (а также AGP) в любых возможных режимах работы архитектуры x86 (включая реальный, защищенный

16-разрядный, защищенный 32-разрядный режимы и режим с линейным адресным пространством). Основное назначение функций PCI BIOS — работа с конфигурационным пространством и генерация специальных циклов шины PCI.

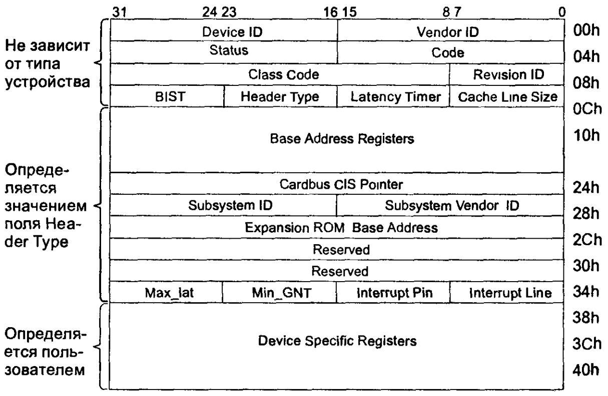


Рис. 3.1. Конфигурационное пространство устройства PCI

Общий вид конфигурационного пространства устройства PCI показан на рис. 3.1. Конфигурационное пространство содержит три области:

- заголовок, не зависящий от типа устройства;
- область, определяемая устройством (значением Header Type);
- область, определяемая пользователем.

С точки зрения программиста важными являются первые две из перечисленных областей, а область пользователя фактически доступна лишь разработчикам устройства, поскольку детальное описание этой области есть только у них и посторонним обычно не предоставляется. Впрочем, чтобы извлечь какую-то полезную информацию из полей, содержащих базовые адреса, также необходимо располагать документацией на устройство: каждый изготовитель трактует их по-своему. Для наглядности достаточно сравнить между собой распределение базовых адресов в видеоконтроллерах 3Dfx [38] и Matrox [75, 76].

Если программисту требуется выполнять с устройством PCI только типовые операции, то ему совершенно не обязательно разрабатывать полноценный драйвер для устройства — для решения некоторых типовых задач достаточно той документации, которую фирмы-изготовители предоставляют через Интернет в режиме свободного доступа. На практике интерес представляют следующие поля конфигурационного пространства:

- Vendor ID — код фирмы-изготовителя устройства;
- Device ID — код устройства;
- Class Code — код класса устройства;
- Base Address Registers — регистры базовых адресов;
- Interrupt Line — номер выделенного устройству прерывания IRQ.

Код изготовителя, код устройства и код класса применяются в процессе поиска заданного устройства. Если необходимо найти конкретное устройство, то поиск выполняется по кодам устройства и его изготовителя (см. вышеупомянутые описания контроллеров Matrox и 3Dfx); если требуется обнаружить все устройства определенного типа, то используется код класса. После того как искомое устройство обнаружено, при помощи регистров базовых адресов можно определить выделенные ему области в адресном пространстве памяти и пространстве ввода-вывода. Наибольший интерес, однако, представляет регистр Interrupt Line, позволяющий выяснить, какая линия IRQ была выделена устройству процедурой PnP BIOS в процессе начальной загрузки компьютера — в некоторых случаях это единственный документированный способ определения номера прерывания.

Функции PCI BIOS

Поскольку конфигурационное пространство не имеет привязки к какой-либо определенной области адресного пространства компьютера, доступ к нему связан с определенными трудностями. С целью упрощения работы с устройствами PCI в BIOS персональных компьютеров были внесены специальные дополнительные функции. Доступ к функциям PCI BIOS при 16-разрядном вызове выполняется через функцию B1h прерывания 1Ah. Для 32-разрядных вызовов используется 32-разрядная точка входа защищенного режима.

Функции PCI BIOS используют регистры процессора для передачи аргументов и получения результатов. При успешном выполнении

функции флаг переноса CF сбрасывается в 0, в случае неудачи — устанавливается в 1.

Прерывание 1Ah, функция B101h: проверить присутствие PCI BIOS в системе

Для проверки присутствия PCI BIOS в системе по прерыванию 1Ah вызывается функция B101h.

Перед вызовом прерывания 1Ah требуется занести в регистр AX код 0B101h.

После выполнения функции в регистрах будут размещены следующие значения:

- в EDX — сигнатура «PCI» («P» — в DL, «C» — в DH и т. д., в старшем байте — пробел);
- в AH — признак присутствия (0 — BIOS присутствует, если в EDX правильная сигнатура; любое другое значение — PCI BIOS отсутствует);
- в AL — аппаратный механизм;
- в BH — номер версии интерфейса PCI (в двоично-десятичном коде);
- в BL — подномер версии интерфейса (в двоично-десятичном коде);
- в CL — номер последней шины PCI в системе (счет номеров начинается с нуля).

Флаг CF также будет содержать признак наличия PCI BIOS (0 — BIOS присутствует, 1 — отсутствует).

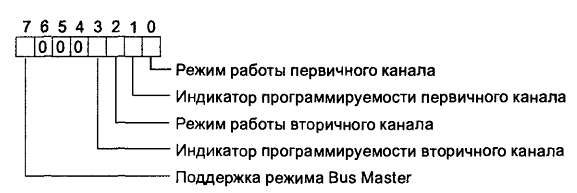


Рис. 3.2. Формат байта-описателя аппаратного механизма шины PCI

На рисунке 3.2 показан формат байта-описателя аппаратного механизма шины: информация, возвращенная в регистре AL, показывает,

какие механизмы функционирования шины PCI реализованы в данной аппаратуре. Спецификация PCI определяет два аппаратных механизма для доступа к конфигурационному пространству. Механизм 1 поддерживается, если установлен бит 0, механизм 2 — если установлен бит 1. Спецификация PCI определяет также механизмы генерации специальных циклов. Бит 4 установлен, если аппаратура может выполнять генерацию специальных циклов на основе механизма 1, бит 5 установлен, если аппаратура может выполнять генерацию специальных циклов на основе механизма 2. Биты 2, 3, 6 и 7 зарезервированы и должны быть равны нулю.

Прерывание 1Ah, функция B102h: найти устройство PCI заданного типа

Для поиска устройства PCI заданного типа по прерыванию 1Ah вызывается функция B102h.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 0B102h;
- в CX — идентификатор устройства (число от 0 до 65535);
- в DX — идентификатор изготовителя (от 0 до 65534);
- в SI — индекс (порядковый номер) устройства заданного типа (от 0 до N).

После выполнения функции в регистрах будут размещены следующие значения:

- в BH — номер шины, к которой подключено устройство (от 0 до 255);
- в BL — номер устройства в старших пяти битах и номер функции в трех младших;
- в AH — код возврата (может принимать значения BAD_ VENDOR_ID, DEVICE_NOT_FOUND и SUCCESFUL);
- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Если необходимо найти все устройства данного типа, то в SI заносится 0. После каждого выполнения функции значение SI надо увеличивать на 1, пока не будет получен код возврата DEVICE_NOT_FOUND (устройство не обнаружено).

Прерывание 1Ah, функция B103h: найти устройство PCI заданного класса

Для поиска устройства PCI заданного класса по прерыванию 1Ah вызывается функция B103h.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 0B103h;
- в ECX — код класса (в младших трех байтах);
- в SI — индекс (порядковый номер) устройства заданного класса (от 0 до N).

После выполнения функции в регистрах будут размещены следующие значения:

- в BH — номер шины, к которой подключено устройство (от 0 до 255);
- в BL — номер устройства в старших пяти битах и номер функции в трех младших;
- в AH — код возврата (может принимать значения DEVICE_NOT_FOUND и SUCCESFUL);
- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Если необходимо найти все устройства данного класса, то в SI заносится 0. После каждого выполнения функции значение SI надо увеличивать на 1, пока не будет получен код возврата DEVICE_NOT_FDUND (устройство не обнаружено).

Прерывание 1Ah, функция B106h: генерировать специальный цикл шины

Для генерации специального цикла шины по прерыванию 1Ah вызывается функция B106h.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 0B106h;
- в BH — номер шины, к которой подключено устройство (от 0 до 255);
- в EDX — данные специального цикла.

После выполнения функции в регистрах будут размещены следующие значения:

- в AH — код возврата (может принимать значения SUCCESFUL и FUNC_NOT_SUPPORTED);
- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Прерывание 1Ah, функция B108h: прочитать байт из конфигурационного пространства заданного устройства

Для считывания байта из конфигурационного пространства заданного устройства по прерыванию 1Ah вызывается функция B108h.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 0B108h;
- в BH — номер шины, к которой подключено устройство (от 0 до 255);
- в BL — номер устройства в старших пяти битах и номер функции в трех младших;
- в DI — порядковый номер байта (от 0 до 255).

После выполнения функции в регистрах будут размещены следующие значения:

- в CL — считанный байт;
- в AH — код возврата (может принимать значения SUCCESFUL и BAD_REGISTER_NUMBER);
- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Прерывание 1Ah, функция B109h: прочитать слово из конфигурационного пространства заданного устройства

Для считывания слова из конфигурационного пространства заданного устройства по прерыванию 1Ah вызывается функция B109h.

Перед вызовом прерывания требуется занести в регистры необходимые значения.

Значения, которые требуется занести в регистры:

- в AX — код 0B109h;
- в BH — номер шины, к которой подключено устройство (от 0 до 255);
- в BL — номер устройства в старших пяти битах и номер функции в трех младших;
- в DI — смещение слова в конфигурационном пространстве (0, 2, 4, ..., 254).

После выполнения функции в регистрах будут размещены следующие значения:

- в CX — считанное слово;
- в AH — код возврата (может принимать значения SUCCESFUL и BAD_REGISTER_NUMBER);
- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Прерывание 1Ah, функция B10Ah: прочитать двойное слово из конфигурационного пространства заданного устройства

Для считывания двойного слова из конфигурационного пространства заданного устройства по прерыванию 1Ah вызывается функция B10Ah.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 0B10Ah;
- в BH — номер шины, к которой подключено устройство (от 0 до 255);
- в BL — номер устройства в старших пяти битах и номер функции в трех младших;
- в DI — смещение двойного слова в конфигурационном пространстве (0, 4, 8, ..., 252).

После выполнения функции в регистрах будут размещены следующие значения:

- в ECX — считанное двойное слово;
- в AH — код возврата (может принимать значения SUCCESFUL и BAD_REGISTER_NUMBER);

- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Прерывание 1Ah, функция B10Bh: записать байт в конфигурационное пространство заданного устройства

Для записи байта в конфигурационное пространство заданного устройства по прерыванию 1Ah вызывается функция B10Bh.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 0B10Bh;
- в BH — номер шины, к которой подключено устройство (от 0 до 255);
- в BL — номер устройства в старших пяти битах и номер функции в трех младших;
- в DI — порядковый номер байта (от 0 до 255);
- в CL — записываемый байт.

После выполнения функции в регистрах будут размещены следующие значения:

- в AH — код возврата (может принимать значения SUCCESFUL и BAD_REGISTER_NUMBER);
- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Прерывание 1Ah, функция B10Ch: записать слово в конфигурационное пространство заданного устройства

Для записи слова в конфигурационное пространство заданного устройства по прерыванию 1Ah вызывается функция B10Ch.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 0B10Ch;
- в BH — номер шины, к которой подключено устройство (от 0 до 255);

- в BL — номер устройства в старших пяти битах и номер функции в трех младших;
- в DI — смещение слова в конфигурационном пространстве (0, 2, 4, ..., 254).
- в CX — записываемое слово.

После выполнения функции в регистрах будут размещены следующие значения:

- в AH — код возврата (может принимать значения SUCCESFUL и BAD_REGISTER_NUMBER);
- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Прерывание 1Ah, функция B10Dh: записать двойное слово в конфигурационное пространство заданного устройства

Для записи двойного слова в конфигурационное пространство заданного устройства по прерыванию 1Ah вызывается функция B10Dh.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 0B10Dh;
- в BH — номер шины, к которой подключено устройство (от 0 до 255);
- в BL — номер устройства в старших пяти битах и номер функции в трех младших;
- в DI — смещение двойного слова в конфигурационном пространстве (0, 4, 8, ..., 252).
- в ECX — записываемое двойное слово.

После выполнения функции в регистрах будут размещены следующие значения:

- в AH — код возврата (может принимать значения SUCCESFUL и BAD_REGISTER_NUMBER, см. табл. 3.1);
- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Прерывание 1Ah, функция B10Eh: получить опции маршрутизации прерываний PCI

Для получения опций маршрутизации прерываний PCI на системной плате и битовой карты прерываний IRQ, выделенных устройствам PCI в исключительное пользование, по прерыванию 1Ah вызывается функция B10Eh.

Перед вызовом прерывания 1Ah требуется занести в регистры следующие значения:

- в AX — код 0B10Eh;
- в BX — значение 0;
- в DS — сегмент или селектор для данных BIOS;
- в ES — сегмент или селектор буфера маршрутизации;
- в DI (при использовании 32-разрядной адресации — в EDI) — смещение буфера маршрутизации.

После выполнения функции в регистрах будут размещены следующие значения:

- в AH — код возврата (может принимать значения SUCCESFUL, BUFFER_TO_SMALL и FUNK_NOT_SUPPORTED);
- в BX — битовая карта прерываний IRQ, выделенных устройствам PCI в исключительное пользование;
- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Адрес, задаваемый сегментом и смещением буфера маршрутизации, должен указывать на структуру следующего вида:

```
typedef struct
    {
        WORD BufferSize;
        BYTE FAR * DataBuffer;
    } IRQRoutingOptionsBuffer;
```

Поля структуры IRQRoutingOptionsBuffer имеют следующее назначение:

- BufferSize — размер буфера данных в байтах;
- DataBuffer — дальний указатель на буфер данных, выделенный для приема информации обо всех встроенных устройствах PCI и всех слотах расширения PCI на системной плате.

После успешного выполнения функции 0B10Eh для каждого размещенного на системной плате устройства PCI и каждого слота

расширения PCI в буфере данных будет размещено по одному 16-байтному элементу. Структура элемента таблицы маршрутизации прерываний PCI показана в табл. 3.1.

Таблица 3.1. Структура элемента таблицы маршрутизации прерываний PCI

Смещение	Размер	Описание
0	BYTE	Номер шины PCI, к которой подключено устройство
1	BYTE	Номер устройства (в старших пяти битах)
2	BYTE	Код соединения для INTA#
3	WORD	Битовая карта IRQ для INTA#
5	BYTE	Код соединения для INTB#
6	WORD	Битовая карта IRQ для INTB#
8	BYTE	Код соединения для INTC#
9	WORD	Битовая карта IRQ для INTC#
11	BYTE	Код соединения для INTD#
12	WORD	Битовая карта IRQ для INTD#
14	BYTE	Номер слота PCI
15	BYTE	Зарезервирован

Каждой линии прерывания, выведенной на слот PCI, в элементе таблицы маршрутизации прерываний PCI соответствует два поля.

- Код соединения позволяет определить, какие линии прерываний соединены между собой проводниками на плате: значение кодов соединения не стандартизировано, но для соединенных между собой линий значение кодов совпадает. Если код соединения равен нулю, то данный контакт не подключен к контроллеру прерываний.
- Битовая карта IRQ показывает, к каким из линий IRQ может быть подсоединена данный контакт разъема. Каждой линии IRQ соответствует один разряд битовой карты (бит 0 — IRQ0, бит 1 — IRQ1 и т. д.): если разряд сброшен в 0, то подключение невозможно, а если установлен в 1 — возможно.

Номер слота, указанный в конце элемента таблицы маршрутизации прерываний, показывает, чему соответствует данный элемент: встроенному устройству или слоту расширения. Для встроенных устройств значение номера слота равно нулю, а нумерация слотов расширения соответствует порядку их размещения на системной

плате, но не обязательно начинается с единицы (при выполнении нумерации могут учитываться слоты ISA).

Прерывание 1Ah, функция B10Fh: присвоить устройству номер прерывания

Данная функция позволяет присвоить устройству определенный номер прерывания IRQ. Функция предназначена для конфигурирования оборудования в процессе загрузки операционной системы и не должна использоваться в драйверах и прикладных программах.

Перед вызовом прерывания 1Ah требуется занести в регистры следующие значения:

- в AX — код 0B10Fh;
- в CL — код контакта на разъеме PCI (INTA# — 0Ah, INTB# — 0Bh, INTC# — 0Ch, INTD# — 0Dh);
- в CH — номер линии IRQ (может принимать значения от 0 до 15);
- в BH — номер шины, к которой подключено устройство (от 0 до 255);
- в BL — номер устройства в старших пяти битах и номер функции в трех младших;
- в DS — сегмент или селектор для данных BIOS.

После выполнения функции в регистрах будут размещены следующие значения:

- в AH — код возврата (может принимать значения SUCCESFUL, SET_FAILED и FUNK_NOT_SUPPORTED);
- в CF — статус возврата (0 — функция успешно выполнена, 1 — ошибка).

Расшифровка значения кодов возврата приведена в табл. 3.2.

Таблица 3.2. Список кодов возврата

Код возврата	Значение в AH	Расшифровка
SUCCESFUL	00h	Операция успешно выполнена
FUNC_NOT_SUPPORTED	81h	Функция не поддерживается данным устройством
BAD_VENDOR_ID	83h	Неверный идентификатор изготовителя
DEVICE_NOT_FOUND	86h	Устройство не найдено

продолжение ⇒

Таблица 3.2 (*продолжение*)

Код возврата	Значение в AH	Расшифроака
BAD_REGISTER_NUMBER	87h	Некорректное значение индекса
SET_FAILED	88h	Операция переустановки номеров прерываний не выполнена
BUFFER_TOO_SMALL	89h	Недостаточный объем буфера

С точки зрения программиста, особый интерес представляют подфункции 01h (проверка наличия PCI BIOS), 02h (найти устройство заданного типа), 03h (найти устройство заданного класса) и 08h–0Ah (прочитать данные из конфигурационного пространства устройства).

Проверять наличие PCI BIOS нужно при запуске программы: если его нет, то нет и возможности использовать относящиеся к нему функции. Проверку наличия в системе нужного устройства и определение его координат на шине PCI (номера шины, номера устройства и функции) также следует выполнять при запуске программы. После обнаружения устройства становится доступной для считывания вся информация из его конфигурационного пространства.

Поиск устройства PCI по коду класса

Поиск устройства определенного типа можно осуществлять по коду класса. Код класса состоит из трех байтов (рис. 3.3): старший байт задает базовый класс (Base Class), средний байт — подкласс (SubClass), младший байт — интерфейс (Interface). В таблице 3.3 приведен перечень базовых классов устройств PCI, а табл. 3.4 содержит полное описание кодов классов.

23 — 16	15 — 8	7 — 0
Базовый класс	Подкласс	Интерфейс

Рис. 3.3. Общая структура поля кода класса

Таблица 3.3. Коды базовых классов

Код класса	Тип устройства
00h	Устройство было изготовлено до введения стандарта на коды классов
01h	Контроллер устройства массовой памяти
02h	Сетевой контроллер
03h	Видеоконтроллер

04h	Устройство мультимедиа
05h	Контроллер памяти
06h	Устройство типа «мост»
07h	Простой коммуникационный контроллер
08h	Основная системная периферия
09h	Устройство ввода
0Ah	Стыковочная станция
0Bh	Процессор
0Ch	Контроллер последовательной шины
0Dh	Контроллер устройства беспроводной передачи данных
0Eh	Интеллектуальный контроллер ввода-вывода
0Fh	Сопутствующий коммуникационный контроллер
10h	Контроллер устройства шифрации/дешифрации
11h	Контроллер устройства сбора и обработки сигналов
12h–FEh	Зарезервированы
FFh	Устройство не подходит ни к одному из перечисленных классов

Таблица 3.4. Полное описание кодов классов устройств

Базовый класс	Подкласс	Интерфейс	Значение
00h	00h	00h	Все устройства, выпущенные до принятия стандарта, за исключением VGA-совместимых
	01h	00h	VGA-совместимые устройства
01h	00h	00h	SCSI-контроллер
	01h	xxh (рис 3 4)	IDE-контроллер
	02h	00h	Контроллер дисковода гибких дисков
	03h	00h	IPI-контроллер
	04h	00h	RAID-контроллер
	80h	00h	Устройство массовой памяти другого типа
02h	00h	00h	Контроллер Ethernet
	01h	00h	Контроллер Token Ring
	02h	00h	Контроллер FDDI
	03h	00h	Контроллер ATM
	04h	00h	Контроллер ISDN
	80h	00h	Сетевой контроллер другого типа

продолжение ⇗

Таблица 3.4 (*продолжение*)

Базовый класс	Подкласс	Интерфейс	Значение
03h	00h	00h	VGA-совместимый контроллер
		01h	8514-совместимый контроллер
	01h	00h	Контроллер XGA
	02h	00h	3D-контроллер
	80h	00h	Другой видеоконтроллер
04h	00h	00h	Видеоустройство
	01h	00h	Аудиоустройство
	02h	00h	Устройство для компьютерной телефонии
	80h	00h	Мультимедиа-устройство другого типа
05h	00h	00h	Контроллер оперативной памяти
	01h	00h	Контроллер Flash-памяти
	80h	00h	Контроллер памяти другого типа
06h	00h	00h	Мост хоста
	01h	00h	Мост ISA
	02h	00h	Мост EISA
	03h	00h	Мост MCA
	04h	00h	Мост PCI-to-PCI
		01h	Мост PCI-to-PCI, поддерживающий вычитающее декодирование
	05h	00h	Мост PCMCIA
	06h	00h	Мост NuBus
	07h	00h	Мост CardBus
	08h	00h	Мост RACEway в режиме прозрачности
		01h	Мост RACEway в режиме оконечного узла
	09h	40h	Полупрозрачный мост PCI-to-PCI, обращенный к хост-процессору «основной» стороной
		80h	Полупрозрачный мост PCI-to-PCI, обращенный к хост-процессору «вторичной» стороной
	80h	00h	Мост другого типа
07h	00h	00h	Стандартный XT-совместимый контроллер последовательного порта

Базовый класс	Подкласс	Интерфейс	Значение
		01h	16450-совместимый контроллер последовательного порта
		02h	16550-совместимый контроллер последовательного порта
		03h	16650-совместимый контроллер последовательного порта
		04h	16750-совместимый контроллер последовательного порта
		05h	16850-совместимый контроллер последовательного порта
		06h	16950-совместимый контроллер последовательного порта
	01h	00h	Параллельный порт
		01h	Двунаправленный параллельный порт
		02h	Параллельный порт типа ECP 1.X
		03h	Контроллер IEEE1284
		FEh	Целевое устройство IEEE1284 (не контроллер)
	02h	00h	Многопортовый последовательный контроллер
	03h	00h	Стандартный модем
		01h	Hayes-совместимый модем, 16450-совместимый интерфейс
		02h	Hayes-совместимый модем, 16550-совместимый интерфейс
		03h	Hayes-совместимый модем, 16650-совместимый интерфейс
		04h	Hayes-совместимый модем, 16750-совместимый интерфейс
	80h	00h	Другое коммуникационное устройство
08h	00h	00h	Стандартный контроллер прерываний типа 8259
		01h	Контроллер прерываний ISA
		02h	Контроллер прерываний EISA
		10h	Контроллер прерываний ввода-вывода APIC
		20h	Контроллер прерываний ввода-вывода APIC

продолжение ⇨

Таблица 3.4 (*продолжение*)

Базовый класс	Подкласс	Интерфейс	Значение
	01h	00h	Стандартный контроллер ПДП типа 8237
		01h	ISA-контроллер ПДП
		02h	EISA-контроллер ПДП
	02h	00h	Стандартный системный таймер типа 8254
		01h	Системный таймер ISA
		02h	Системные таймеры EISA (два таймера)
	03h	00h	Стандартный контроллер часов реального времени
		01h	Контроллер часов реального времени ISA
	04h	00h	Стандартный контроллер «горячего подключения» PCI
	80h	00h	Системная периферия другого типа
09h	00h	00h	Контроллер клавиатуры
	01h	00h	Контроллер дигитайзера
	02h	00h	Контроллер мыши
	03h	00h	Контроллер сканера
	04h	00h	Стандартный контроллер игрового порта
		01h	Контроллер игрового порта с программируемым интерфейсом
	80h	00h	Контроллер другого устройства ввода данных
0Ah	00h	00h	Стандартная станция подключения
	80h	00h	Нестандартная станция подключения
0Bh	00h	00h	Процессор типа 386
	01h	00h	Процессор типа 486
	02h	00h	Процессор типа Pentium
	10h	00h	Процессор типа Alpha
	20h	00h	Процессор типа Power PC
	30h	00h	Процессор типа MIPS
	40h	00h	Сопроцессор
0Ch	00h	00h	IEEE1394 (FireWire)

Базовый класс	Подкласс	Интерфейс	Значение
		10h	IEEE1394, следующий за спецификацией 1394 OpenHCI
	01h	00h	ACCESS.bus
	02h	00h	SSA
	03h	00h	Устройство USB, следующее за спецификацией Universal Host Controller
		10h	Устройство USB, следующее за спецификацией Open Host Controller
		80h	Устройство USB без определенного программного интерфейса
		FEh	Устройство USB (не хост-контроллер)
	04h	00h	Fibre Channel
	05h	00h	SMBus
0Dh	00h	00h	IRDA-совместимый контроллер
	01h	00h	IR-контроллер потребителя
	10h	00h	RF-контроллер
	80h	00h	Другой тип контроллерв беспроводной связи
0Eh	00h	xxh	Интеллектуальный контроллер ввода-вывода (I2O) с архитектурой, соответствующей спецификации 1.0
		00h	Буфер сообщений FIFO со смещением 040h
0Fh	01h	00h	TV-контроллер
	02h	00h	Аудиоконтроллер
	03h	00h	Голосовой контроллер
	04h	00h	Контроллер данных
10h	00h	00h	Сетевой или компьютерный шифратор/дешифратор
	10h	00h	Игровой шифратор/дешифратор
	80h	00h	Шифратор/дешифратор другого типа
11h	00h	00h	DPIO-модуль
	80h	00h	Контроллер устройства сбора и обработки сигналов другого типа

В программе PCITest, приведенной в листинге 3.1, поиск VGA-совместимого видеоконтроллера осуществляется по коду класса, присвоенному устройствам такого типа (030000h). Обнаружив первое устройство с соответствующим кодом (в системе, вообще говоря, может быть несколько видеоконтроллеров), программа прекращает поиск и выводит на экран значения некоторых полей конфигурационного пространства. Для запуска данного примера пригоден любой компьютер с шиной PCI и PCI- или AGP-видеоконтроллером.



Рис. 3.4. Формат байта интерфейса кода класса IDE-контроллера

Листинг 3.1. Поиск видеоконтроллера, определение кода изготовителя, используемого адресного пространства и номера прерывания IRQ

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Параметры устройства PCI
VendorID      DW ? ;идентификатор изготовителя
DeviceID      DW ? ;идентификатор устройства
```

```
ClassCode     DD ? ;тип устройства
BaseAddress0  DD ? ;базовый адрес 0
BaseAddress1  DD ? ;базовый адрес 1
BaseAddress2  DD ? ;базовый адрес 2
BaseAddress3  DD ? ;базовый адрес 3
BaseAddress4  DD ? ;базовый адрес 4
BaseAddress5  DD ? ;базовый адрес 5
InterruptLine DB ? ;номер используемого прерывания IRQ
; Координаты устройства PCI
BusNumber     DB ? ;номер шины
DeviceNumber  DB ? ;номер устройства и номер функции
; Счетчик операций нажатия/отпускания клавиш
PressCounter  DW ?
; Текстовые сообщения
PCI   DB 0,25,"ТЕСТИРОВАНИЕ ФУНКЦИЙ PCI BIOS",0
      DB 4,26,"ПАРАМЕТРЫ ВИДЕОКОНТРОЛЛЕРА",0
      DB 6,28,"Номер шины:",0
      DB 7,22,"Номер устройства:",0
      DB 8,25,"Номер функции:",0
      DB 9,12,"Идентификатор изготовителя:",0
      DB 10,14,"Идентификатор устройства:",0
      DB 11,24,"Тип устройства:",0
      DB 12,23,"Базовый адрес 0:",0
      DB 13,23,"Базовый адрес 1:",0
      DB 14,23,"Базовый адрес 2:",0
      DB 15,23,"Базовый адрес 3:",0
      DB 16,23,"Базовый адрес 4:",0
      DB 17,23,"Базовый адрес 5:",0
      DB 18,8,"Номер используемого прерывания:",0
NoIRQ DB 18,40,"не используется",0
AnyK  DB YELLOW,24,29,"Нажмите любую клавишу",0
; Сообщения об ошибках
NoPCI  DB 12,18,"Система не поддерживает PCI BIOS",0
NoSVGA DB 12,23,"Видеоконтроллер SVGA PCI не найден",0
BadReg DB 12,28,"Неверный номер регистра",0
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC PCITest
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
```

продолжение ⇩

Листинг 3.1 (*продолжение*)

```
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Установить зеленый цвет символов и черный фон
        mov     [TextColorAndBackground],LIGHTGREEN
; Вывести текстовые сообщения на экран
        MShowText 15,PCI
; Установить желтый цвет символов и черный фон
        mov     [TextColorAndBackground],YELLOW

; Опрос PCI-устройств
; Проверить наличие PCI BIOS
        mov     AX,0B101h
        int     1Ah
        jc      @@PCIBIOSNotFound
        cmp     EDX,20494350h
        jne     @@PCIBIOSNotFound
; Найти видеоконтроллер (первое устройство
; типа 30000h)
        mov     AX,0B103h
        mov     ECX,030000h
        mov     SI,0
        int     1Ah
        jc      @@DeviceNotFound
        mov     [BusNumber],BH
        mov     [DeviceNumber],BL
; Получить идентификатор изготовителя
        mov     AX,0B109h ;читать слово
        mov     DI,0      ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [VendorID],CX
; Получить идентификатор устройства
        mov     AX,0B109h ;читать слово
        mov     DI,2      ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [DeviceID],CX
; Получить тип устройства (самопроверка)
        mov     AX,0B10Ah ;читать двойное слово
        mov     DI,8      ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        shr     ECX,8
        mov     [ClassCode],ECX
; Получить базовый адрес 0
        mov     AX,0B10Ah ;читать двойное слово
        mov     DI,10h    ;смещение слова
```

```
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [BaseAddress0],ECX
; Получить базовый адрес 1
        mov     AX,0B10Ah ;читать двойное слово
        mov     DI,14h    ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [BaseAddress1],ECX
; Получить базовый адрес 2
        mov     AX,0B10Ah ;читать двойное слово
        mov     DI,18h    ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [BaseAddress2],ECX
; Получить базовый адрес 3
        mov     AX,0B10Ah ;читать двойное слово
        mov     DI,1Ch    ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [BaseAddress3],ECX
; Получить базовый адрес 4
        mov     AX,0B10Ah ;читать двойное слово
        mov     DI,20h    ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [BaseAddress4],ECX
; Получить базовый адрес 5
        mov     AX,0B10Ah ;читать двойное слово
        mov     DI,24h    ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [BaseAddress5],ECX

; Получить номер используемого устройством
; прерывания IRQ
        mov     AX,0B108h ;читать байт
        mov     DI,3Ch    ;смещение байта
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [InterruptLine],CL

; Вывести полученные данные на экран в
; шестнадцатеричном коде
        ; Отобразить основные параметры устройства
        MShowHexByte  6, 40, [BusNumber]
        mov     AH,[DeviceNumber]
        shr     AH,3
        MShowHexByte  7, 40, AH
        mov     AH,[DeviceNumber]
        and     AH,111b
```

продолжение ⇗

Листинг 3.1 (*продолжение*)

```
        MShowHexByte   8, 40, AH
        MShowHexWord   9, 40, [VendorID]
        MShowHexWord  10, 40, [DeviceID]
        MShowHexDWord 11, 40, [ClassCode]
        ; Отобразить базовые адреса
        MShowHexDWord 12, 40, [BaseAddress0]
        MShowHexDWord 13, 40, [BaseAddress1]
        MShowHexDWord 14, 40, [BaseAddress2]
        MShowHexDWord 15, 40, [BaseAddress3]
        MShowHexDWord 16, 40, [BaseAddress4]
        MShowHexDWord 17, 40, [BaseAddress5]
        ; Прерывание используется?
        cmp     [InterruptLine],0FFh
        je      @@IRQNotUsed
        ; Вывести номер используемого прерывания IRQ
        MShowHexByte  18, 40, [InterruptLine]
        jmp short @@End
@@IRQNotUsed:
        MShowString NoIRQ
; Окончание работы
@@End:  MShowColorString AnyK
        call    GetChar
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h

; Обработка ошибок
@@BadRegisterNumber:
        MFatalError BadReg ;неверный номер регистра
@@DeviceNotFound:
        MFatalError NoSVGA ;не найден видеоконтроллер
@@PCIBIOSNotFound:
        MFatalError NoPCI  ;не поддерживается PCI BIOS
ENDP PCITest
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"

END
```

Вызов функций PCI BIOS в защищенном режиме

Когда процессор x86 находится в защищенном режиме, способ вызова функций PCI BIOS зависит от используемого режима адресации.

Если используется 16-разрядная внутрисегментная адресация для кода и данных, то функции PCI BIOS можно вызвать при помощи прерывания Int 1Ah независимо от того, в каком режиме работает процессор — реальном, защищенном или виртуальном.

Обращение к функциям PCI BIOS может также производиться через специальную точку вызова с физическим адресом 000FFE6Eh, которая имитирует прерывание Int 1Ah (при работе в защищенном режиме для CS используется база селектора 0F000h). Обращение по физическому адресу обеспечивает прямой доступ к функциям PCI BIOS, то есть позволяет обойти любые драйверы, которые могут перехватывать прерывание Int 1Ah.

При работе в 32-разрядном защищенном режиме для доступа к функциям PCI BIOS могут использоваться средства сервиса BIOS32, однако поддержка функций BIOS в 32-разрядном режиме для персональных компьютеров не является обязательной. Процедура проверки наличия такой поддержки предполагает обращение к физическим адресам памяти, поэтому обычно производится из реального режима, до переключения в защищенный режим. Если 32-разрядный режим поддерживается, то в области памяти BIOS, расположенной в диапазоне 0E0000h-0FFFFFh, должна присутствовать специальная 16-байтная структура данных —. Служебный каталог (BIOS32 Service Directory). Структура каталога показана в табл. 3.5.

Таблица 3.5. Структура Служебного каталога PCI BIOS32

Смещение	Размер	Назначение
00h	4 байта	Сигнатура служебного каталога в коде ASCII: «_32_»
04h	DWORD	32-разрядный физический адрес точки входа BIOS32
08h	BYTE	Номер версии реализации BIOS32 (имеет значение 00h)
09h	BYTE	Размер Служебного каталога BIOS32 в 16-байтных параграфах (имеет значение 01h)
0Ah	BYTE	Контрольная сумма (дополнение суммы байтов 0–9 до нуля)
0Bh	5 байт	Зарезервированы, имеют значение 0

Процесс поиска Служебного каталога BIOS32 заключается в сканировании памяти ПЗУ BIOS: производится поиск сигнатуры «_32_» в диапазоне 0E0000h-0FFFFFh по 16-байтным параграфам (начало Служебного каталога выровнено на границу 16 байт). После

обнаружения сигнатуры производится вычисление и проверка контрольной суммы: если сумма совпадает, то Служебный каталог найден.

Чтобы получить параметры, необходимые для доступа к PCI BIOS в 32-разрядном режиме, требуется выполнить дальний вызов через точку входа BIOS32. Перед выполнением вызова требуется записать в регистры следующую информацию:

- в EAX — идентификатор запрашиваемого сервиса, который для PCI BIOS имеет значение «$PCI» (049435024h);
- в EBX — селектор функции, значение которого должно быть равно нулю.

После выполнения вызова в регистре AL будет возвращен код результата (0 — операция успешно завершена, 80h — некорректный идентификатор сервиса, 81h — недопустимое значение селектора функции).

Если вызов завершился успешно, в регистрах процессора будет размещена следующая информация:

- в EBX — физический адрес базы сервиса BIOS;
- в ECX — размер сегмента сервиса BIOS;
- в EDX — точка входа в сервис BIOS (смещение относительно базы, возвращенной в EBX).

Параметры, возвращенные в регистрах EBX, ECX и EDX, можно использовать при выполнении дальнего вызова (физический адрес базы и размер сегмента используются для создания дескрипторов, загружаемых в регистры CS и DS).

ПРИМЕЧАНИЕ

Для программ, использующих функции PCI BIOS, рекомендуется задавать размер стека не менее 1024 байт. При вызове функций в защищенном режиме уровень привилегий должен быть достаточным для того, чтобы подпрограммы имели доступ к пространству ввода-вывода и механизму управления прерываниями.

Глава 4
Видеоконтроллеры

Развитие аппаратного обеспечения AT-совместимых персональных компьютеров напоминает рост кораллового рифа: новые компоненты оборудования устанавливаются поверх старых, которые постепенно выходят из употребления (отмирают). Такой способ проектирования имеет один существенный недостаток: архитектура системы все больше и больше захламляется устаревшими, практически не используемыми компонентами. Особенно ясно захламленность системы видна на примере видеоконтроллеров: в каждой новой конструкции сохраняются компоненты всех предшествующих.

С целью увеличения скорости вывода изображения на экран приходится работать непосредственно с аппаратурой контроллера, и программисты вынуждены разбираться со множеством хитростей и трюков, которые были использованы разработчиками оборудования. Чтобы не перегружать читателей бесполезной информацией, в данной главе мы рассмотрим только такие свойства и особенности видеоконтроллеров, которые необходимо учитывать в процессе программирования.

Основные типы графических режимов

Современные видеоконтроллеры поддерживают разнообразные текстовые и графические режимы, различающиеся разрешением, палитрой используемых цветов, частотой кадровой развертки и т. д.

Текстовые режимы различают по разрешению (числу отображаемых символов по горизонтали и вертикали) и цветовой палитре (возможен монохромный или 16-цветный режим).

Для графических режимов основным признаком классификации является количество одновременно отображаемых цветов и, соответственно, количество двоичных разрядов, необходимых для хранения одной точки изображения. Различают следующие типы графических режимов:

- монохромный (1-битное кодирование цвета точки);
- 4-цветный CGA (2-битное кодирование);
- 16-цветный EGA/VGA (4-битное кодирование);
- 256-цветный SVGA (8-битное кодирование);
- HiColor (16-битное кодирование);
- TrueColor (24-битное или 32-битное кодирование).

При этом принципы работы с видеопамятью для каждого из основных типов видеорежимов настолько отличаются друг от друга, что в современных видеоконтроллерах используется для каждого из этих типов отдельный аппаратный блок. В настоящее время старые режимы (начиная от монохромных и кончая 16-цветными) вышли из употребления: вывод информации в этих режимах осуществляется очень медленно (вследствие неудачной реализации механизма работы с памятью). Поэтому ниже будут рассматриваться только текстовый 16-цветный режим и современные графические режимы.

Прежде чем можно будет начать работу в каком-либо режиме, необходимо вначале перевести систему в этот режим. Однако сделать это не так-то просто. Категорически не рекомендуется начинающим программистам ставить эксперименты по прямому управлению режимами работы видеоконтроллера через его регистры: некорректная установка параметров может вывести из строя монитор. Особенно велик риск для распространенных в нашей стране удешевленных моделей мониторов: снижение цены достигается за счет упрощения конструкции, в частности — ослабления защиты.

Для видеоконтроллеров разделение операций на аппаратные и программные является абсолютно жестким: любые переключения режимов должны выполняться при помощи функций BIOS видеоконтроллера или фирменных драйверов, а выводить информацию нужно напрямую в видеопамять. Переключать видеорежимы только с помощью «родного» программного обеспечения самого контроллера приходится по причине слабой стандартизированности устройств этого типа: с тех пор, как фирма IBM потеряла контроль над рынком персональных компьютеров, разработчики периферийного оборудования в буквальном смысле творят, что хотят. Даже внутри одной фирмы могут существовать несколько разных групп

устройств (линий), в каждой из которых применяется собственная внутренняя архитектура. С большим трудом ассоциация VESA сумела стандартизировать функции управления режимами работы видеоконтроллеров.

Устанавливать необходимый видеорежим следует один раз при запуске прикладной программы на весь период ее выполнения. Переключать режимы в процессе работы крайне нежелательно: при этом наблюдаются разнообразные видеоэффекты, которые могут сильно раздражать пользователя (сбой синхронизации, «снег», мерцание и т. д.). Даже такая безобидная операция, как полное гашение экрана монитора на время переключения видеорежима, может быть неприятной для оператора: в течение нескольких секунд он будет не в состоянии контролировать работу системы.

Следовательно, видеорежим должен позволять выполнять все требуемые в программе операции — если требуется вывод и текстовой, и графической информации, то нужно сразу установить графический режим. Существует два основных способа программной установки видеорежима, которые мы рассмотрим ниже: с помощью функций VGA BIOS и с помощью функций VESA BIOS.

Функции VGA BIOS

Графические режимы VGA за прошедшее с момента создания (1987 год) этого типа видеоконтроллеров время сильно устарели, а текстовые используются двадцать лет и продолжают успешно применяться (благодаря простоте выполнения операций в этих режимах). Поэтому в настоящее время интерес для программистов представляет только небольшая подгруппа функций из стандартного набора VGA BIOS, предназначенная для работы в текстовом режиме:

- установка режима;
- управление положением и размером курсора;
- переключение страниц;
- управление шрифтами.

Для вызова функций VGA BIOS используется прерывание `Int 10h`. Набор функций очень большой, но в целом — устаревший. Ниже рассматриваются только те функции, применение которых до сих пор является целесообразным. Более подробную информацию о функциях можно получить в книгах [3, 10, 29, 33, 40].

Прерывание Int 10h, функция 00h: установить видеорежим

Функция предназначена для установки заданного видеорежима. Коды VGA-режимов приведены в табл. 4.1. Все современные видеоконтроллеры поддерживают текстовый режим, а также сохраняют совместимость с устаревшими графическими режимами VGA на тот случай, если пользователю понадобится запустить какую-нибудь из старых программ.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AH — значение 00h;
- в AL — код видеорежима.

Таблица 4.1. Коды VGA-совместимых режимов

Код режима	Тип режима	Разрешение	Число цветов	Адрес буфера
00h	Текстовый цветной*	40×25	16	B8000h
01h	Текстовый цветной	40×25	16	B8000h
02h	Текстовый цветной*	80×25	16	B8000h
03h	Текстовый цветной	80×25	16	B8000h
04h	Графический цветной	320×200	4	B8000h
05h	Графический цветной*	320×200	4	B8000h
06h	Графический цветной	640×200	2	B8000h
07h	Текстовый монохромный	80×25	2	B0000h
0Dh	Графический цветной	320×200	16	A0000h
0Eh	Графический цветной	640×200	16	A0000h
0Fh	Графический монохромный	640×350	2	A0000h
10h	Графический цветной	640×350	16	A0000h
11h	Графический монохромный	640×480	2	A0000h
12h	Графический цветной	640×480	16	A0000h
13h	Графический цветной	320×200	256	A0000h

Знак «*» означает, что в данном режиме отключен механизм преобразования палитры.

Функцию установки текстового режима обычно нужно вызывать при входе в программу, работающую в этом режиме (поскольку, вообще говоря, неизвестно, в каком режиме работала DOS перед запуском программы). Функцию также вызывают перед возвратом в DOS из программы, использовавшей какой-либо графический режим —

чтобы не было проблем с запуском последующих программ. Мы неоднократно пользовались данной функцией в примерах, приведенных в предыдущих главах.

Фрагмент программного кода, в котором выполняется установка текстового режима, выглядит следующим образом:

```
mov     AH,0   ;код функция установки режима
mov     AL,03h ;код режима
int     10h    ;вызов прерывания
```

Прерывание Int 10h, функция 01h: установить размер курсора

Функция предназначена для задания размера курсора в текстовом видеорежиме. Применяется она обычно в текстовых редакторах для уведомления пользователя о текущем режиме работы (вставка или замещение символов).

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AH — код 01h;
- в CH — номер начальной строки курсора в знакоместе;
- в CL — номер конечной строки курсора в знакоместе.

Номера начальной и конечной строк курсора задаются относительно верхней границы знакоместа. Для задания номеров начальной и конечной строк в регистрах CL и CH используются только разряды 0–4, в разряды 5–7 требуется записать нули.

ПРИМЕЧАНИЕ

Курсор можно сделать невидимым, если установить номер строки текущей позиции курсора за пределами нижней границы экрана (при помощи функции 02h). Это — самый надежный способ; существуют и другие приемы, но далеко не на всех видеоконтроллерах они дают нужный результат.

Прерывание Int 10h, функция 02h: установить позицию курсора

Функция позиционирует курсор на экране.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AH — код 02h;
- в BH — номер текстовой видеостраницы;

- в DH — номер строки (Y);
- в DL — номер столбца (X).

ПРИМЕЧАНИЕ

За начало системы координат принят левый верхний угол экрана, причем ось X направлена слева направо, а ось Y — сверху вниз В стандартном текстовом режиме значение X можно задавать в пределах от 0 до 79, значение Y — от 0 до 24. Если задать значение Y = 25, то курсор становится невидимым (уходит за нижнюю границу видеостраницы).

Прерывание Int 10h, функция 03h: получить позицию и размер курсора

Функция определяет текущую позицию курсора на заданной видеостранице, а также размер курсора.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AH — код 03h;
- в BH — номер видеостраницы.

После выполнения функции в регистрах будут размещены следующие значения:

- в CH — номер начальной строки курсора в знакоместе;
- в CL — номер конечной строки курсора в знакоместе;
- в DH — номер строки текущей позиции курсора;
- в DL — номер столбца текущей позиции курсора.

Прерывание Int 10h, функция 05h: установить видеостраницу

В текстовом режиме данная функция позволяет выбрать в качестве текущей (рабочей) одну из нескольких доступных видеостраниц. Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AH — код 05h;
- в AL — номер видеостраницы, которую требуется установить.

Прерывание Int 10h, функция 10h, подфункция 00h: установить один регистр палитры

Функция предназначена для 16-цветных режимов (текстовых и графических). Она позволяет переопределить цвет, соответствующий одному из кодов цвета.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1000h;
- в BL — номер регистра палитры (0–15);
- в BH — цвет.

Прерывание Int 10h, функция 10h, подфункция 01h: установить цвет рамки экрана

Функция позволяет изменить цвет рамки экрана (по умолчанию он черный).

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1001h;
- в BH — цвет.

Прерывание Int 10h, функция 10h, подфункция 02h: установить все регистры палитры

Функция предназначена для 16-цветных режимов. Она позволяет переопределить значения 16 регистров палитры и регистра рамки экрана.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1002h;
- в ES:DX — указатель на массив из 17 байт (байты 0–15 содержат цвета палитры, байт 16 — цвет рамки).

Прерывание Int 10h, функция 10h, подфункция 03h: переключить бит атрибута «мерцание/яркость»

Функция переключает значение седьмого бита байта атрибутов символа: мерцание или яркий фон.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1003h;
- в BL — значение бита 7 в байте атрибутов (0 — яркий фон, 1 — мерцание символа).

ПРИМЕЧАНИЕ

Функция воздействует на режим отображения всех символов, а не на один конкретный символ. По умолчанию установка бита 7 в байте атрибутов символа вызывает его мерцание.

Ни мерцание, ни яркий фон не следует применять без особой необходимости — они утомляют оператора, сидящего за экраном монитора.

Прерывание Int 10h, функция 10h, подфункция 07h: прочитать один регистр палитры

Функция предназначена для 16-цветных режимов. Она позволяет прочитать значение цвета из заданного регистра палитры.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1007h;
- в BL — номер регистра палитры (0–15).

После выполнения функция возвращает значение цвета в BH.

Прерывание Int 10h, функция 10h, подфункция 08h: прочитать один регистр палитры

Функция позволяет прочитать значение цвета рамки экрана из соответствующего регистра видеоконтроллера.

Перед вызовом прерывания требуется занести:

в регистр AX код 100Bh.

После выполнения функция возвращает значение цвета в регистре BH.

Прерывание Int 10h, функция 10h, подфункция 09h: прочитать все регистры палитры

Функция предназначена для 16-цветных режимов. Она позволяет прочитать и сохранить в оперативной памяти значения 16 регистров палитры и регистра рамки экрана.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1009h;
- в ES:DX — указатель на массив из 17 байт для сохранения кодов цвета.

После выполнения функции байты 0–15 указанного массива содержат цвета палитры, байт 16 — цвет рамки.

Прерывание Int 10h, функция 10h, подфункция 10h: установить один регистр ЦАП

Функция предназначена для 256-цветных режимов. Она позволяет изменить оттенок, соответствующий одному из кодов цвета, путем перезаписи значений интенсивностей красного, зеленого и синего цветов в соответствующем регистре цифро-аналогового преобразователя.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1010h;
- в BX — номер регистра ЦАП (0–255);
- в DH — новое значение интенсивности красного цвета (0–63);
- в CH — новое значение интенсивности зеленого цвета (0–63);
- в CL — новое значение интенсивности синего цвета (0–63).

Прерывание Int 10h, функция 10h, подфункция 12h: перезагрузить группу регистров ЦАП

Функция предназначена для 256-цветных режимов. Она позволяет изменить набор оттенков, соответствующих группе цветовых кодов, путем перезаписи значений интенсивностей красного, зеленого и синего цветов в последовательно расположенных регистрах цифро-аналогового преобразователя.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1012h;
- в BX — номер начального регистра ЦАП (0–255);
- в CX — число перезагружаемых регистров (1–256);
- в ES:DX — указатель на массив загружаемых оттенков из 3×CX байт.

Массив оттенков должен содержать число 3-байтных строк, равное значению, записанному в CX. Первый байт в строке кодирует интенсивность красного, второй — зеленого, третий — синего цветов.

Прерывание Int 10h, функция 10h, подфункция 15h: прочитать один регистр ЦАП

Функция предназначена для 256-цветных режимов. Она позволяет прочитать содержимое заданного регистра цифро-аналогового преобразователя.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1015h;
- в BX — номер регистра ЦАП (0–255).

После завершения функции в регистрах будут размещены следующие значения:

- в DH — значение интенсивности красного цвета;
- в CH — значение интенсивности зеленого цвета;
- в CL — значение интенсивности синего цвета.

Прерывание Int 10h, функция 10h, подфункция 17h: прочитать группу регистров ЦАП

Функция предназначена для 256-цветных режимов. Она позволяет сохранить в оперативной памяти компьютера набор оттенков, соответствующих группе цветовых кодов.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1017h;
- в BX — номер начального регистра ЦАП (0–255);
- в CX — число считываемых регистров (1–256);
- в ES:DX — указатель на область памяти для сохранения считанной информации (массив размером 3×CX байт).

После выполнения функции массив оттенков будет содержать 3-байтные строки: первый байт в строке кодирует интенсивность красного, второй — зеленого, третий — синего цветов.

Прерывание Int 10h, функция 11h, подфункция 00h: загрузить шрифт пользователя для текстового видеорежима

Функция обеспечивает загрузку заданного пользователем шрифта в знакогенератор. Применяется в текстовом видеорежиме.

ПРИМЕЧАНИЕ

Обычно весь шрифт загружается целиком (CX = 256, DX = 0), однако при необходимости возможна перезапись отдельного участка в наборе символов (в CX записывается число заменяемых символов, в DX — номер первого символа в заменяемом участке). Память знакогенератора может содержать до восьми наборов шрифтов, однако обычно используется только блок с нулевым номером (BL = 0). В цветном текстовом режиме VGA используется шрифт 8×16, то есть BH = 16. Символы представлены в растровой форме, каждой точке изображения соответствует один бит в маске символа, а каждой строке — один байт; символ кодируется 16 байтами, а полная таблица шрифта занимает 4 Кбайт.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 1100h;
- в BH — число байтов в матрице символа;

- в BL — номер загружаемого блока знакогенератора;
- в CX — число загружаемых символов;
- в DX — номер первого загружаемого символа в таблице кодов;
- в ES:BP — указатель на таблицу, содержащую маски символов загружаемого шрифта.

Функции VESA BIOS

Программистам приходится иметь дело с прерываниями VESA BIOS в тех случаях, когда система функционирует под управлением любой типовой однозадачной (например, MS-DOS) или самодельной многозадачной (например, Linux) операционной системы, так как фирмы-изготовители поставляют драйверы только для Windows.

Стандарт VESA унифицировал некоторые наиболее важные операции при работе с видеоконтроллером — установку видеорежимов и управление видеопамятью. Начиная с версии 2.0, данный стандарт поддерживает работу с линейным буфером видеопамяти. Полное описание стандарта VESA 3.0 на английском языке [96] можно свободно загрузить из Интернета с сервера ассоциации VESA (www.vesa.org).

Обращение к VESA BIOS выполняется по прерыванию 10h с номером функции 4Fh. После выполнения вызова в регистре AX будет возвращен код результата (статус возврата). В AL будет находиться основной код возврата: если AL = 4Fh — вызов успешно выполнен, если AL ≠ 4Fh — вызванная функция не поддерживается данной версией BIOS. В AH будет записан код, поясняющий результат:

- 0 — вызов функции успешно выполнен;
- 1 — вызов не выполнен;
- 2 — функция не поддерживается в данной аппаратной конфигурации;
- 3 — вызов функции невозможен в данном видеорежиме.

Рассмотрим некоторые наиболее полезные функции VESA BIOS 2.0, которые поддерживаются распространенными в настоящее время моделями видеоконтроллеров (в новой редакции стандарта VESA 3.0 изменения в эти функции не вносились).

Прерывание Int 10h, функция 4Fh, подфункция 00h: получить информацию о версии VESA BIOS

Функция предназначена для считывания информации о версии VESA BIOS (информация представлена в виде структуры, описанной в табл. 4.2).

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 4F00h;
- в ES:DI — адрес буфера объемом 512 байт для сохранения информации о версии VESA BIOS (информация сохраняется в виде структуры, описанной в табл. 4.2).

Таблица 4.2. Формат информации VESA BIOS

Смещение	Размер	Описание
00h	4 байта	Сигнатура 'VESA'
04h	WORD	Номер версии VESA (0200h или 0300h)
06h	DWORD	Указатель на строку OEM (с наименованием микросхемы видеоконтроллера)
0Ah	4 байта	Возможности графического контроллера: бит 0: 0 — 6-разрядный ЦАП, 1 — 8-разрядный; бит 1: 0 — контроллер VGA-совместимый, 1 — несовместимый; бит 2: 0 — обычная работа с RAMDAC, 1 — при программировании больших блоков информации в RAMDAC использовать бит очистки функции 09h; биты 3–31 зарезервированы
0Eh	DWORD	Указатель на список видеорежимов, поддерживаемых VESA и OEM
12h	WORD	Число 64-килобайтных блоков памяти (объем видеопамяти, деленный на 64 Кбайт)
14h	WORD	Код версии программной реализации VESA
16h	DWORD	Указатель на строку с названием фирмы-изготовителя
1Ah	DWORD	Указатель на строку с названием изделия
1Eh	DWORD	Указатель на строку с кодом версии изделия
22h	222 байта	Зарезервировано для последующих версий
100h	256 байт	Область данных для строк OEM

Подфункцию 00h необходимо вызвать перед началом использования других VESA-функций. Она позволяет определить, имеется ли у видеоконтроллера поддержка команд VESA, и если да — то какой версии. Чтобы имелась возможность использования линейного буфера, версия VESA должна быть не ниже 2.0.

Вначале нужно убедиться, что функция была выполнена: после вызова функции в регистре AL должен находиться код 4Fh. Затем следует проверить наличие сигнатуры 'VESA' в четырех первых байтах возвращенного функцией блока данных, а также убедиться, что байт со смещением 05h от начала блока (старший байт номера версии) содержит значение не меньше 2.

Прерывание Int 10h, функция 4Fh, подфункция 01h: получить информацию о параметрах видеорежима

Функция позволяет проверить наличие в данной версии BIOS режима с заданным номером и определить его свойства.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 4F01h;
- в CX — код видеорежима, параметры которого нужно определить;
- в ES:DI — адрес буфера объемом 256 байт для сохранения информации о видеорежиме (табл. 4.3).

Таблица 4.3. Формат информации о видеорежиме

Смещение	Размер	Описание
00h	WORD	Атрибуты режима: бит 0: 0 — режим не поддерживается, 1 — режим поддерживается; бит 1: всегда 1 (зарезервирован); бит 2: 0 — TTY функции не поддерживаются, 1 — TTY функции поддерживаются; бит 3: 0 — монохромный режим, 1 — цветной режим; бит 4: 0 — текстовый режим, 1 — графический режим; бит 5: 0 — VGA-совместимый режим, 1 — VGA-несовместимый режим;

Смещение	Размер	Описание
		бит 6: 0 — режим оконной адресации видеопамяти поддерживается, 1 — не поддерживается;
		бит 7: 0 — режим линейной адресации видеопамяти не поддерживается, 1 — поддерживается;
		биты 8–15 зарезервированы
02h	BYTE	Атрибуты окна А:
		бит 0: 0 — окно неперемещаемое, 1 — поддерживается перемещение окна;
		бит 1: 0 — чтение из окна запрещено, 1 — разрешено;
		бит 2: 0 — запись в окно запрещена, 1 — разрешена;
		биты 3–7 зарезервированы
03h	BYTE	Атрибуты окна В (аналогично байту атрибутов окна А)
04h	WORD	Гранулярность (дробность) окна (наименьшая величина в килобайтах, на которую можно переместить окно)
06h	WORD	Размер окна в килобайтах
08h	WORD	Начальный сегмент окна А в адресном пространстве процессора
0Ah	WORD	Начальный сегмент окна В
0Ch	DWORD	FAR-адрес функции позиционирования окна (она эквивалентна функции 4Fh с подфункцией 05h)
10h	WORD	Число байтов, которое приходится на одну строку развертки
12h	WORD	Горизонтальное разрешение экрана в пикселах (в графическом режиме) или знакоместах (в текстовом режиме)
14h	WORD	Вертикальное разрешение экрана в пикселах или знакоместах
16h	BYTE	Ширина знакоместа в пикселах
17h	BYTE	Высота знакоместа в пикселах
18h	BYTE	Число плоскостей памяти
19h	BYTE	Число битов на пиксел
1Ah	BYTE	Число банков памяти
1Bh	BYTE	Тип модели памяти:
		00h — текстовый режим;
		01h — графический режим CGA;
		02h — графический режим Hercules;
		03h — планарный режим;

продолжение ⇨

Таблица 4.3 (*продолжение*)

Смещение	Размер	Описание
		04h — режим упакованных пикселов;
		05h — нецепочечный режим № 4, 256 цветов;
		06h — режим Direct Color (HiColor или TrueColor);
		07h — режим YUV;
		08h–0Fh — зарезервировано для дополнений VESA;
		10h–FFh — зарезервировано для дополнений OEM
1Ch	BYTE	Размер банка памяти в килобайтах
1Dh	BYTE	Число полных видеостраниц (экранов), помещающихся в видеопамяти, минус единица
1Eh	BYTE	Зарезервирован
Поля Direct Color		
1Fh	BYTE	Размер компоненты красного цвета в битах
20h	BYTE	Начальная битовая позиция компоненты красного цвета
21h	BYTE	Размер компоненты зеленого цвета в битах
22h	BYTE	Начальная битовая позиция компоненты зеленого цвета
23h	BYTE	Размер компоненты синего цвета в битах
24h	BYTE	Начальная битовая позиция компоненты синего цвета
25h	BYTE	Размер резервной компоненты в битах
26h	BYTE	Начальная битовая позиция резервной компоненты
27h	BYTE	Атрибуты режима Direct Color
Поля, присутствующие только в VBE 2.0 и позднейших реализациях стандарта		
28h	DWORD	Физический (абсолютный) 32-разрядный адрес буфера кадра
2Ch	DWORD	Указатель на начало заэкранной памяти, то есть на начало второй видеостраницы (введен в версии 2.0, но изъят из версии 3.0; теперь данное поле считается зарезервированным и содержит значение 0)
30h	WORD	Объем заэкранной памяти в килобайтах (введен в версии 2.0, но изъят из версии 3.0; теперь данное поле считается зарезервированным и содержит значение 0)

Смещение	Размер	Описание
Поля, присутствующие только в VBE 3.0 и позднейших реализациях стандарта		
32h	WORD	Число байтов, которое приходится на одну строку развертки в линейном режиме
34h	BYTE	Количество видеостраниц в страничном режиме
35h	BYTE	Количество видеостраниц в линейном режиме
36h	BYTE	Размер компоненты красного цвета в битах в линейном режиме
37h	BYTE	Начальная битовая позиция компоненты красного цвета в линейном режиме
38h	BYTE	Размер компоненты зеленого цвета в битах в линейном режиме
39h	BYTE	Начальная битовая позиция компоненты зеленого цвета в линейном режиме
40h	BYTE	Размер компоненты синего цвета в битах в линейном режиме
41h	BYTE	Начальная битовая позиция компоненты синего цвета в линейном режиме
42h	BYTE	Размер резервной компоненты в битах в линейном режиме
43h	BYTE	Начальная битовая позиция резервной компоненты в линейном режиме
44h	DWORD	Максимальная частота вывода пикселов в графических режимах (в Гц)
48h	184 байта	Зарезервировано

Подфункцию 01h обычно вызывают перед установкой нового видеорежима, чтобы удостовериться в его реализации в данном видеоконтроллере и в возможности использования линейной адресации видеопамяти в этом режиме. После вызова функции в регистре AL должен находиться код 4Fh, в AH — код 0. После этого нужно проверить седьмой бит слова атрибутов режима (линейная адресация поддерживается, если он равен 1), а затем прочитать адрес линейного видеобуфера из 32-разрядного (двойного) слова со смещением 28h от начала структуры данных.

Подфункции 00h и 01h необходимо использовать, если нет полной уверенности в том, что видеоконтроллер поддерживает используемый в вашей программе видеорежим. В программе VESA_BIOS_Test, приведенной в листинге 4.1, используются указанные подфункции, чтобы проверить наличие и версию VESA BIOS, а затем поочередно отобразить на экран параметры всех имеющихся видеорежимов.

Основной модуль программы функционирует в текстовом режиме и использует процедуры вывода данных на экран, описанные в главе 1 «Работа с клавиатурой», процедуры перевода чисел, описанные в главе 2 «Недокументированные возможности процессоров Intel 80x86», и вспомогательную процедуру ShowVESAString для вывода текстовых строк из области данных VESA BIOS.

Листинг 4.1. Проверка наличия VESA BIOS, а также режима с линейной адресацией, палитрой 256 цветов, разрешением 640×480 и получение его параметров

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Код исследуемого видеорежима
GraphicsMode DW ?
; Место для хранения информации VESA BIOS
VESA_BIOS DB 512 DUP(?)
; Место для хранения информации о параметрах видеорежима
VESA_info DB 256 DUP(?)
; Текстовые сообщения
Txt0 DB LIGHTCYAN,0,33,"TECT VESA BIOS",0
     DB YELLOW,24,29,"Нажмите любую клавишу",0
Txt1 DB 2,29,"Обнаружен драйвер VESA",0
     DB 4,29,"Сигнатура:",0
     DB 5,21,"Номер версии VESA:",0
     DB 6,31,"Имя OEM:",0
     DB 7,22,"Объем памяти, Мб:",0
     DB 12,25,"Номера поддерживаемых режимов:",0
Txt2 DB 4,27,"Параметры режима №     h:",0
     DB 6,13,"Атрибуты режима:",0
     DB 7,13,"Атрибуты окна A:",0
     DB 8,13,"Атрибуты окна B:",0
     DB 9,2,"Гранулярность окна (Кбайт):",0
     DB 10,9,"Размер окна (Кбайт):",0
     DB 11,4,"Начальный сегмент окна A:",0
     DB 12,4,"Начальный сегмент окна B:",0
     DB 13,5,"Адрес функции поз. окна:",0
     DB 14,2,"Байтов на строку развертки:",0
     DB 15,3,"Горизонтальное разрешение:",0
```

```
    DB 16,5,"Вертикальное разрешение:",0
    DB 17,11,"Ширина знакоместа:",0
    DB 18,11,"Высота знакоместа:",0
    DB 19,5,"Число плоскостей памяти:",0
    DB 6,46,"Число битов на пиксел:",0
    DB 7,49,"Число банков памяти:",0
    DB 8,51,"Тип модели памяти:",0
    DB 9,44,"Размер банка памяти (Кб):",0
    DB 10,43,"Число полных видеостраниц:",0
    DB 11,43,"Размер красной компоненты:",0
    DB 12,40,"Нач. поз. красной компоненты:",0
    DB 13,43,"Размер зеленой компоненты:",0
    DB 14,40,"Нач. поз. зеленой компоненты:",0
    DB 15,45,"Размер синей компоненты:",0
    DB 16,42,"Нач. поз. синей компоненты:",0
    DB 17,41,"Размер резервной компоненты:",0
    DB 18,43,"Нач. поз. рез. компоненты:",0
    DB 19,41,"Линейный адрес буфера кадра:",0
Err1 DB 25,0,"Команды VESA не поддерживаются",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC VESA_BIOS_Test
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести заголовок
        MShowColorText 2,Txt0

; ПРОВЕРИТЬ НАЛИЧИЕ VESA BIOS
; Получить информацию о наличии VESA BIOS
        mov     AX,DS
        mov     ES,AX
        mov     AX,4F00h
        mov     DI,offset VESA_BIOS
        int     10h
```

продолжение ⇩

Листинг 4.1 (*продолжение*)

```
; Проверить результат выполнения функции
        cmp     AX,4Fh
        jne     @@Err
; Вывести поясняющий текст
        mov     [TextColorAndBackground],LIGHTGREEN
        MShowText 6,Txt1
        mov     [TextColorAndBackground],WHITE
; Вывести сигнатуру
        mov     SI,offset VESA_BIOS
        MShowASCIIField 4,40,0,4
; Вывести номер версии
        MShowHexWord 5,40,<[word ptr VESA_BIOS+4]>
; Вывести имя OEM
        call    ShowVESAString
; Вывести объем памяти
        mov     AX,[word ptr VESA_BIOS+12h]
        shr     AX,4  ;разделить на 16
        MShowDecWord 7,40,AX
; Вывести список номеров режимов
        mov     [ScreenString],14
        mov     [ScreenColumn],0
        mov     BX,[word ptr VESA_BIOS+0Eh]
        mov     AX,[word ptr VESA_BIOS+10h]
        mov     ES,AX
        mov     CX,128
@@j0:   mov     AX,[ES:BX]
        cmp     AX,0FFFFh
        je      @@j1
        inc     [ScreenColumn]
        call    ShowHexWord
        add     BX,2
        loop    @@j0
; Ожидать нажатия любой клавиши
@@j1:   call    GetChar

; ПОКАЗАТЬ ПАРАМТРЫ ДОСТУПНЫХ ВИДЕОРЕЖИМОВ
; Загрузить адрес массива номеров
        mov     BX,[word ptr VESA_BIOS+0Eh]
@@NextMode:
; Загрузить кодовый номер очередного режииа
        mov     AX,[word ptr VESA_BIOS+10h]
        mov     ES,AX
        mov     AX,[ES:BX]
        mov     [GraphicsMode],AX
        cmp     AX,0FFFFh
        je      @@End
; Очистить экран
        call    ClearScreen
; Вывести заголовок
```

```
        MShowColorText 2,Txt0
; Получить параметры видеорежима
        mov     AX,DS
        mov     ES,AX
        mov     AX,4F01h
        mov     CX,[GraphicsMode]
        mov     DI,offset VESA_info
        int     10h
; Установить зеленый цвет символов и черный фон
        mov     [TextColorAndBackground],LIGHTGREEN
; Вывести текстовые сообщения на экран
        MShowText 29,Txt2
; Вывести номер режима
        MShowHexWord   4,46,[GraphicsMode]
; Установить белый цвет символов и черный фон
        mov     [TextColorAndBackground],WHITE
; Вывести значения полей на экран
        ; Атрибуты режима
        MShowHexWord   6,30,<[word ptr VESA_info]>
        ; Атрибуты окна A
        MShowHexByte   7,30,[VESA_info+02h]
        ; Атрибуты окна B
        MShowHexByte   8,30,[VESA_info+03h]
        ; Гранулярность окна
        MShowDecWord   9,30,<[word ptr VESA_info+04h]>
        ; Размер окна
        MShowDecWord  10,30,<[word ptr VESA_info+06h]>
        ; Начальный сегмент окна A
        MShowHexWord  11,30,<[word ptr VESA_info+08h]>
        ; Начальный сегмент окна B
        MShowHexWord  12,30,<[word ptr VESA_info+0Ah]>
        ; Адрес функции позиционирования окна
        MShowHexDWord 13,30,<[dword ptr VESA_info+0Ch]>
        ; Байт на одну строку развертки
        MShowDecWord  14,30,<[word ptr VESA_info+10h]>
        ; Горизонтальное разрешение
        MShowDecWord  15,30,<[word ptr VESA_info+12h]>
        ; Вертикальное разрешение
        MShowDecWord  16,30,<[word ptr VESA_info+14h]>
        ; Ширина знакоместа
        MShowDecByte  17,30,[VESA_info+16h]
        ; Высота знакоместа
        MShowDecByte  18,30,[VESA_info+17h]
        ; Число плоскостей памяти
        MShowDecByte  19,30,[VESA_info+1Bh]
        ; Число битов на пиксел
        MShowDecByte   6,70,[VESA_info+19h]
        ; Число банков памяти
        MShowDecByte   7,70,[VESA_info+1Ah]
        ; Тип модели памяти
```

продолжение ⇗

Листинг 4.1 (*продолжение*)

```
        MShowDecByte   8,70,[VESA_info+1Bh]
        ; Размер банка памяти
        MShowDecByte   9,70,[VESA_info+1Ch]
        ; Число полных видеостраниц
        MShowDecByte  10,70,[VESA_info+1Dh]
        ; Размер красной компоненты
        MShowDecByte  11,70,[VESA_info+1Fh]
        ; Начальная позиция красной компоненты
        MShowDecByte  12,70,[VESA_info+20h]
        ; Размер зеленой компоненты
        MShowDecByte  13,70,[VESA_info+21h]
        ; Начальная позиция зеленой компоненты
        MShowDecByte  14,70,[VESA_info+22h]
        ; Размер синей компоненты
        MShowDecByte  15,70,[VESA_info+23h]
        ; Начальная позиция синей компоненты
        MShowDecByte  16,70,[VESA_info+24h]
        ; Размер резервной компоненты
        MShowDecByte  17,70,[VESA_info+25h]
        ; Начальная позиция резервной компоненты
        MShowDecByte  18,70,[VESA_info+26h]
        ; Линейный адрес буфера кадра
        MShowHexDWord 19,70,<[dword ptr VESA_info+28h]>
        call    GetChar
        add     BX,2
        jmp     @@NextMode
; КОНЕЦ ПРОГРАММЫ
@@End:  ; Установить текстовый режим
        mov     AX,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h

; СООБЩЕНИЯ ОБ ОШИБКАХ
@@Err:  MFatalError Errl
ENDP VESA_BIOS_Test

;***********************************************
;* ВЫВОД СТРОКИ ИЗ ОБЛАСТИ ДАННЫХ VESA НА ЭКРАН *
;***********************************************
PROC ShowVESAString near
        pusha
        push    ES
; Настраить регистр ES на видеопамять
        mov     AX,0B800h
        mov     ES,AX
        cld
```

```
; Вычислить адрес строки в видеопамяти
        mov     DI,6*160+40*2
; Загрузить атрибут цвета в AH
        mov     AH,[TextColorAndBackground]
; Установить указатель на строку
        push    DS
        mov     BX,offset VESA_BIOS + 06h
        mov     SI,[BX]
        mov     BX,[BX+2]
        mov     DS,BX
        ; Вывести строку
@@L1:   lodsb
        and     AL,AL
        jz      @@L2
        stosw
        jmp     @@L1
@@L2:   pop     DS
        pop     ES
        popa
        ret
ENDP ShowVESAString
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить "математические" процедуры для перевода
; чисел из двоичного кода в десятичный
include "list2_05.inc"

END
```

ВНИМАНИЕ

Поскольку программа из листинга 4.1 не переключает видеоконтроллер в заданный режим, а только проверяет его наличие, запускать тест можно на компьютере любой конфигурации. Однако для получения информации о свойствах VESA-режимов нужно, чтобы видеоконтроллер имел встроенную или программную (в виде специальной программы-драйвера) поддержку VESA BIOS.

Прерывание Int 10h, функция 4Fh, подфункция 02h: установить видеорежим с заданным номером

Функция устанавливает видеорежим с заданным номером и режим адресации видеопамяти (чтобы использовать линейную адресацию, необходимо установить бит 14 в коде режима).

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 4F02h;
- в BX — код видеорежима.

Код режима имеет следующий формат:

- биты 0–8 — номер режима (если бит 8 равен 0, то это режим производителя видеоконтроллера, если 1 — VESA-режим);
- биты 9–13 — зарезервированы для дальнейших расширений и равны 0;
- бит 14 — тип режима адресации видеопамяти: 0 — страничный режим (память адресуется участками по 64 Кбайт), 1 — линейный режим;
- бит 15 — признак очистки видеопамяти при переключении режимов (0 — очищать память, 1 — не очищать).

Стандартизированные номера режимов приведены в табл. 4.4. Обозначения «K» и «M» в графе «Число цветов» представляют собой множители: $K = 2^{10} = 1024$, $M = 2^{20} = 1\,048\,576$.

При работе с функцией установки видеорежима нужно учитывать следующее.

- Режимы с номерами меньше 100h определяются изготовителем видеоконтроллера и не обрабатываются VESA-функциями. Если функции установки видеорежима будет послан код со значением меньше 100h, то она передаст его на обработку прерыванию BIOS 10h.
- Видеоконтроллеры различных производителей могут интерпретировать режимы TrueColor с номерами 10Fh, 112h, 115h, 118h, 11Bh как 24- или как 32-битовые, то есть данные режимы определены неоднозначно. Для уточнения числа байтов, выделенных на одну точку, нужно вызвать функцию 01h, получить информацию о видеорежиме и проверить размер резервной компоненты (байт со смещением 25h): если 0 — 24-битовый режим, если 8 — 32-битовый.
- Коды режимов с разрешением более 1280×1024 не стандартизированы, и их приходится определять по таблице доступных режимов, заданной изготовителем видеоконтроллера.
- Режим 81FFh служит для обеспечения доступа ко всей имеющейся видеопамяти без изменения ее содержимого.

Таблица 4.4. Стандартизированные коды VESA-видеорежимов

Номер режима	Тип	Разрешение	Число цветоа	Бит на точку
100h	Графический	640×400	256	8
101h	Графический	640×480	256	8
102h	Графический	800×600	16	4
103h	Графический	800×600	256	8
104h	Графический	1024×768	16	4
105h	Графический	1024×768	256	8
106h	Графический	1280×1024	16	4
107h	Графический	1280×1024	256	8
108h	Текстовый	80×60	16	—
109h	Текстовый	132×25	16	—
10Ah	Текстовый	132×43	16	—
10Bh	Текстовый	132×50	16	—
10Ch	Текстовый	132×60	16	—
10Dh	Графический	320×200	32 K	1:5:5:5
10Eh	Графический	320×200	64 K	5:6:5
10Fh	Графический	320×200	16 M	8:8:8 или 8:8:8:8
110h	Графический	640×480	32 K	1:5:5:5
111h	Графический	640×480	64 K	5:6:5
112h	Графический	640×480	16 M	8:8:8 или 8:8:8:8
113h	Графический	800×600	32 K	1:5:5:5
114h	Графический	800×600	64 K	5:6:5
115h	Графический	800×600	16 M	8:8:8 или 8:8:8:8
116h	Графический	1024×768	32 K	1:5:5:5
117h	Графический	1024×768	64 K	5:6:5
118h	Графический	1024×768	16 M	8:8:8 или 8:8:8:8
119h	Графический	1280×1024	32 K	1:5:5:5
11Ah	Графический	1280×1024	64 K	5:6:5
11Bh	Графический	1280×1024	16 M	8:8:8 или 8:8:8:8
81FFh	Специальный	—	—	—

Прерывание Int 10h, функция 4Fh, подфункция 03h: определить код текущего видеорежима

Функция возвращает код текущего видеорежима.

Перед вызовом прерывания требуется занести в регистр AX код 4F03h.

После выполнения функции в регистрах будут размещены следующие значения:

- в AX — код результата выполнения функции (код возврата);
- в BX — код текущего видеорежима (см. табл. 4.4).

Прерывание Int 10h, функция 4Fh, подфункция 04h: сохранить или восстановить состояние видеоконтроллера

Функция сохраняет или восстанавливает режим работы видеоконтроллера. Сохранение состояния заключается том, что функция записывает в выделенный буфер оперативной памяти содержимое всех регистров видеоконтроллера. Содержимое видеопамяти функция не записывает — эту операцию необходимо выполнять отдельно.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 4F04h;
- в DL — код подфункции (0 — получить размер буфера, необходимого для сохранения состояния, 1 — сохранить состояние, 2 — восстановить состояние);
- в CX — флаги, уточняющие, состояние каких именно блоков видеоконтроллера нужно сохранить или восстановить (бит 0 — состояние аппаратуры, бит 1 — состояние данных BIOS, бит 2 — состояние DAC, бит 3 — состояние регистров, биты 4–15 зарезервированы);
- в ES:BX — указатель на буфер (нужен только в том случае, когда в DL не 0).

После выполнения функции в регистр AX будет помещен код возврата, в BX — число блоков по 64 байта, необходимых для хранения состояния (если в DL 0).

Подфункция 04h позволяет запомнить состояние видеоконтроллера перед переходом в какой-либо другой режим, чтобы потом можно было с ее помощью вернуться в исходное состояние.

Прерывание Int 10h, функция 4Fh, подфункция 05h: управление окнами видеопамяти

Функция предназначена для управления отображением видеопамяти на адресное пространство процессора при использовании страничного доступа (через окно 64 Кбайт).

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 4F05h;
- в BH — код выполняемой операции (0 — установить окно видеопамяти, 1 — получить номер текущего окна видеопамяти);
- в BL — номер окна (0 — окно A, 1 — окно B);
- в DX — номер окна в видеопамяти (в единицах дробности); задается только при BH = 0.

После выполнения функции в регистр AX будет помещен код возврата, в DX — адрес окна в видеопамяти в единицах дробности (выдается только при BH = 1).

Подфункция 05h применяется в оконном режиме адресации, когда в каждый момент времени процессору доступен только один участок (сегмент) видеопамяти размером в 64 Кбайт. Ее необходимо вызывать каждый раз, когда требуется перейти к работе с другим участком.

Прерывание Int 10h, функция 4Fh, подфункция 06h: получить или установить длину логической строки развертки

Функция читает текущую длину логической строки либо устанавливает ее новое значение.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 4F06h;
- в BL — код выполняемой операции (0 — установить длину строки развертки в пикселах, 1 — получить текущую длину строки

развертки в пикселах, 2 — установить длину строки развертки в байтах, 3 — получить максимально возможную длину строки развертки в пикселах;

- в CX — требуемую ширину строки (задается только для операций 0 и 2: при BL = 0 — в пикселах, при BL = 2 — в байтах).

После выполнения функции в регистрах будут размещены следующие значения:

- в AX — код возврата;
- в BX — число байт на строку развертки;
- в CX — число пикселов на строку развертки (округленное до ближайшего целого пиксела);
- в DX — максимальное число строк развертки.

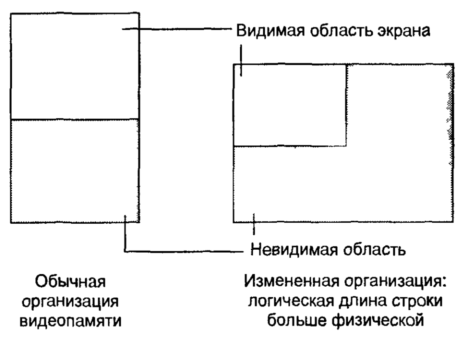


Рис. 4.1. Реорганизация видеопамяти после изменения логической длины строки

Подфункция 06h используется для изменения организации видеопамяти с целью упрощения адресации (путем выравнивания длины строки на величину, равную 2^N —1024 или 2048). Видеоконтроллеры позволяют делать длину строки в видеопамяти (логическую) больше реальной (физической) длины строки экрана за счет реорганизации неиспользуемых областей памяти. Обычно невидимая область располагается под видимой (рис. 4.1, *слева*), однако после увеличения логической длины строки происходит перераспределение памяти — появляется невидимый участок справа, а высота нижнего неотображаемого участка уменьшается (рис. 4.1, *справа*).

Выравнивание логической длины строки на 2^N позволяет:

- упростить вычисление адресов памяти (используя сдвиги и логические операции вместо умножения и сложения);
- реализовать (при необходимости) плавную прокрутку изображения по вертикали аппаратными средствами видеоконтроллера;
- существенно упростить контроль границ сегментов при оконном режиме адресации видеопамяти.

Для пояснения последнего пункта рассмотрим обычную организацию видеопамяти в режиме оконной адресации. Если длина строки в пикселах не кратна 2^N, то сегмент будет содержать нецелое число экранных строк, и границы сегментов не будут совпадать с границами экрана (рис. 4.2, *слева*). В результате при выводе изображения на экран необходимо контролировать не только нижнюю, но и правую границу сегмента (вывод выполняется по точкам, слева направо и сверху вниз). Такую операцию контроля приходится включать внутрь всех циклов вывода изображения, что сильно тормозит работу: вывод точки на экран выполняется одной простой командой пересылки данных MOV, и включение любых других команд (в том числе для проверки границ) в цикл вывода точек замедляет его выполнение в несколько раз. Замедление особенно заметно на современных быстрых видеоконтроллерах, у которых скорость записи в память измеряется десятками мегабайтов в секунду.



Рис. 4.2. Выравнивание границ сегментов после изменения логической длины строки

Если выровнять длину строки в пикселах на 2^N, то сегмент будет состоять из целого числа экранных строк, а начало каждого сегмента будет совпадать с началом строки экрана (рис. 4.2, *справа*). Операция контроля пересечения границы очередного сегмента при этом проводится во внешнем цикле вывода строки изображения, а не во внутреннем цикле рисования точек. Объем памяти при низких раз-

решениях (при длине строки 640 или 800 точек) обычно не является препятствием для операции выравнивания даже для видеоконтроллеров нижнего ценового уровня, так как самые примитивные современные карты имеют память емкостью 4–8 Мбайт.

Прерывание Int 10h, функция 4Fh, подфункция 07h: получить или установить координаты левого верхнего угла экрана

Функция читает или устанавливает номер пиксела, отображаемого в левом верхнем углу экрана.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 4F07h;
- в BH — 0 (регистр зарезервирован для последующих расширений);
- в BL — код выполняемой операции (00h — установить начало экрана, 01h — получить начало экрана, 80h — установить начало экрана в процессе обратного хода луча по кадру);
- CX — первый отображаемый пиксел на строке развертки (задается только при BL = 00h или 80h);
- DX — первая отображаемая строка развертки (задается только при BL = 00h или 80h).

После выполнения функции в регистрах будут размещены следующие значения:

- в AX — код возврата;
- в CX — первый отображаемый пиксел на строке развертки (только при BL = 01h);
- в DX — первая отображаемая строка развертки (только при BL = 01h).

Подфункция 07h имеет две основные области применения: прокрутка экрана и переключение экранов. В обоих случаях происходит перемещение по видеопамяти области, отображаемой на экран, но в первом случае — плавно, во втором — скачкообразно. Такое перемещение возможно благодаря тому, что система управления памятью видеоконтроллера позволяет произвольно выбирать точку, с которой начинается вывод на экран (позицию левого верхнего угла экрана относительно видеопамяти).

Для реализации прокрутки изображения влево необходимо перед выводом очередного кадра прибавлять единицу к номеру начальной точки, а для реализации прокрутки вправо – вычитать единицу. Для реализации прокрутки вверх необходимо к номеру начальной точки прибавлять число точек в строке, для реализации прокрутки вниз – вычитать его.

Для выполнения переключения экранов (окон) необходимо вычислить смещение второго окна относительно начала видеопамяти. Оно равно числу точек в строке изображения, помноженному на число строк на экране и на число байтов, приходящихся на одну точку. Далее для каждого нового кадра происходит поочередная установка номера первого выводимого пиксела (0 или величина смещения второго окна). Пока одно окно выводится на экран, происходит перезапись информации в другом окне. Таким образом можно исключить эффекты типа «снега» и разрезания изображения, возникающие при одновременной записи в участок видеопамяти и выводе на экран информации из этого же участка.

Прерывание Int 10h, функция 4Fh, подфункция 08h: получить или изменить формат регистров палитры

Функция читает или устанавливает разрядность регистров палитры (6 или 8 бит) Основное назначение функции – расширение цветовой палитры в режиме 256 цветов с 2^{18} до 2^{24} оттенков В настоящий момент функция устарела, так как для работы с широкой цветовой палитрой гораздо эффективнее использовать режим TrueColor. Единственной областью ее применения, вероятно, являются графические редакторы, работающие с форматом файлов PCX.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 4F08h;
- в BL — код выполняемой операции (0 — установить формат регистров палитры, 1 — получить формат регистров палитры);
- в BH (при BL = 0) — требуемая разрядность регистров ЦАП в битах (допустимы только значения 6 и 8).

ПРИМЕЧАНИЕ

Функция выдает код ошибки 3 при вызове в режимах DirectColor или YUV.

После выполнения функции в регистрах будут размещены следующие значения:

- в AX — код возврата;
- в BH — текущая разрядность регистров ЦАП в битах.

Прерывание Int 10h, функция 4Fh, подфункция 09h: сохранить или изменить содержимое регистров ЦАП

Функция считывает или устанавливает содержимое таблицы регистров цифро-аналогового преобразователя (DAC), которая ставит в соответствие кодам цвета реальные значения по красному, зеленому и синему компонентам (в 16- и 256-цветных режимах). Это доработанный вариант подфункций 12h и 17h функции 10h прерывания Int 10h VGA BIOS. Функция 09h позволяет перепрограммировать таблицу палитры (в режиме 256 цветов) с целью получения набора оттенков, более подходящего для решения стоящих перед программистом задач, чем стандартный набор. Например, можно организовать плавные цветовые переходы.

Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 4F09h;
- в BL — код выполняемой операции (00h — установить палитру, 01h — прочитать (сохранить) палитру, 02h — установить дополнительную (вторичную) палитру, 03h — прочитать дополнительную (вторичную) палитру, 80h — установить палитру в процессе обратного хода луча по кадру);
- в CX — число загружаемых (или сохраняемых) регистров палитры (максимум 256);
- в DX — номер первого из загружаемых (сохраняемых) регистров;
- в ES:DI — указатель на таблицу, из которой загружается или в которой сохраняется содержимое регистров палитры.

Формат строки таблицы палитры следующий: байт выравнивания (нулевой), байт интенсивности красного цвета, байт интенсивности зеленого цвета, байт интенсивности синего цвета.

Прерывание Int 10h, функция 4Fh, подфункция 0Ah: получить таблицу доступа к интерфейсу защищенного режима

Функция возвращает указатель на таблицу доступа к интерфейсу 32-разрядного защищенного режима. Перед вызовом прерывания требуется занести в регистры следующие значения:

- в AX — код 4F0Ah;
- в BX — значение 00h.

После выполнения функции в регистрах будут размещены следующие значения:

- в AX — код возврата;
- в ES:DI — указатель на таблицу доступа к интерфейсу защищенного режима;
- в CX — размер таблицы доступа в байтах, включая размер кода для работы в защищенном режиме.

Таблица доступа к интерфейсу защищенного режима содержит четыре слова-указателя, вслед за которыми располагается программный код для работы в защищенном режиме. Указатели размещены в таблице доступа в следующем порядке:

- слово со смещением 00h содержит смещение кода функции 05h (для защищенного режима) от начала таблицы доступа;
- слово со смещением 02h содержит смещение кода функции 07h (для защищенного режима) от начала таблицы доступа;
- слово со смещением 04h содержит смещение кода функции 09h (для защищенного режима) от начала таблицы доступа;
- слово со смещением 06h содержит смещение таблицы портов и описателей области памяти, для доступа к которым нужно устанавливать соответствующий уровень привилегий, от начала таблицы доступа (значение 0000h в данном слове указывает на отсутствие таблицы).

Таблица портов и областей памяти устроена следующим образом: в начале находится список 16-разрядных адресов портов ввода вывода, который заканчивается терминатором FFFFh. За терминатором списка портов следуют 32-разрядный базовый адрес области памяти, 16-разрядный лимит сегмента и терминатор описателя области памяти FFFFh.

Для обеспечения доступа к программному коду в защищенном режиме нужно либо скопировать все содержимое таблицы доступа в 32-разрядный сегмент, либо настроить селекторы и права ввода-вывода прямо на соответствующую область ПЗУ.

Регистры видеоконтроллера

Необходимость непосредственной работы с регистрами видеоконтроллера реально возникает только после перехода в защищенный режим работы процессора Intel x86, когда недоступны видеофункции BIOS.

Управление режимом работы внешнего устройства путем прямой записи информации в его регистры — анахронизм, и довольно опасный. Установка режима, не поддерживаемого подключенным к контроллеру монитором, или запись ошибочного значения в один из регистров может привести к поломке оборудования (контроллера или монитора). Все операции переключения режимов может и должен выполнять собственный процессор контроллера, а не центральный процессор компьютера. Однако, пока изготовители вынуждены придерживаться допотопной технологии управления (для обеспечения совместимости с VGA), необходимо помнить правила работы с регистрами видеоконтроллера.

Все регистры видеоконтроллера являются восьмиразрядными. В состав стандартного контроллера VGA-типа входит шесть групп управляющих регистров:

- внешние регистры;
- регистры контроллера электронно-лучевой трубки (ЭЛТ);
- регистры синхронизатора;
- регистры графического контроллера;
- регистры контроллера атрибутов;
- регистры цифро-аналогового преобразователя VGA (ЦАП VGA).

В настоящее время, вообще говоря, изделия различных изготовителей совершенно несовместимы друг с другом на уровне регистров. Однако современные видеоконтроллеры имитируют работу контроллера VGA с целью сохранения совместимости со старым программным обеспечением. В частности, допускается использование некоторых (не всех) управляющих регистров по тем же правилам, какие были приняты для VGA (причем не только в старых, но и в новых

видеорежимах). Ниже будут рассматриваться только такие регистры, а более подробные сведения о регистрах VGA можно при необходимости найти в литературе [1, 29, 33].

Внешние регистры

Большая часть регистров видеоконтроллера доступна только в режиме индексной адресации. Особенность внешних регистров заключается в том, что все они адресуются напрямую, то есть каждый имеет свой собственный адрес порта.

В группу внешних регистров входят:

- регистр управления различными режимами работы видеоконтроллера (Miscellaneous Output Register), который доступен только для записи через порт 3C2h;
- регистр управления линиями FC0 и FC1 EGA (Feature Control Register), доступный только для записи через порт 3BAh в монохромном режиме и через порт 3DAh — в цветном;
- нулевой регистр состояния видеоконтроллера (Input Status Register Zero), доступный только для считывания через порт 3C2h;
- первый регистр состояния видеоконтроллера (Input Status Register One), доступный для считывания через порт 3BAh в монохромном режиме и через порт 3DAh — в цветном;
- регистр разрешения работы видеоподсистемы (Video Subsystem Enable Register), доступный для записи и считывания через порт 3C3h.

С точки зрения программиста интерес в этой группе представляет только первый регистр состояния видеоконтроллера. Разряды данного регистра имеют следующее значение:

- бит 0 — признак обратного хода луча по строке (0 — обратный ход луча, 1 — прямой ход);
- биты 1 и 2 — не используются;
- бит 3 — признак обратного хода луча по кадру (0 — прямой ход, 1 — обратный ход луча);
- биты 4–7 — не используются.

Бит 3 применяется в компьютерных играх для обеспечения синхронизации работы программного обеспечения и видеоконтроллера (для предотвращения эффекта разрезания изображения), а также при выполнении перезагрузки регистров палитры цифро-аналогового

преобразователя (которая должна выполняться во время обратного хода луча). Остальные биты нигде не используются.

Регистры синхронизатора

Каждая группа регистров VGA (за исключением внешних регистров) имеет свой адресный регистр. В него записывается индекс безадресного регистра, к которому выполняется обращение. Порт безадресного регистра в пространстве ввода-вывода размещается всегда сразу после порта адресного регистра, поэтому возможна как раздельная запись информации (вначале в адресный регистр заносится индекс, затем выполняется обращение к соответствующему безадресному порту), так и одновременная запись (в адресный порт выводится слово, младший байт которого служит индексом, а старший содержит записываемое значение).

Адресный регистр синхронизатора доступен для записи и считывания через порт 3C4h. После занесения в него значения индекса можно работать с соответствующим безадресным регистром через порт 3C5h. Значения индексов регистров синхронизатора приведены в табл. 4.5.

Таблица 4.5. Регистры синхронизатора

Индекс	Регистр
0	Регистр сброса (Reset Register)
1	Регистр режима синхронизации (Clocking Mode Register)
2	Регистр маскирования памяти (Map Mask Register)
3	Регистр выбора таблицы символов (Character Map Select Register)
4	Регистр режима памяти (Memory Mode Register)

Регистр сброса

Регистр используется для сброса синхронизатора. Разряды регистра имеют следующее значение:

- бит 0 — асинхронный сброс (установка значения этого разряда в 0 вызывает асинхронный сброс синхронизатора и потерю хранящейся в видеопамяти информации; для нормальной работы видеоконтроллера разряд должен быть установлен в 1);
- бит 1 — синхронный сброс (установка значения этого разряда в 0 вызывает синхронный сброс синхронизатора; при работе контроллера разряд должен быть установлен в 1);
- биты 2–7 — не используются.

Разряд синхронного сброса (бит 1) должен быть установлен в 0 перед началом записи информации в регистр режима синхронизации и возвращен в 1 после завершения этой операции.

Регистр режима синхронизации

Этот регистр позволяет управлять временными циклами синхронизатора. Текстовые и современные графические режимы видеоконтроллера не требуют выполнения каких-либо операций с регистром режима синхронизации и значение в нем лучше не изменять, чтобы не повредить монитор.

Регистр маскирования памяти

Регистр маскирования памяти позволяет запретить запись информации в отдельные цветовые слои (плоскости, битовые матрицы) видеопамяти. Разряды регистра имеют следующее назначение:

- бит 0 — разрешение записи в слой 0 (Синий);
- бит 1 — разрешение записи в слой 0 (Зеленый);
- бит 2 — разрешение записи в слой 0 (Красный);
- бит 3 — разрешение записи в слой 0 (Яркость);
- биты 4–7 — не используются.

В графических 16-цветных режимах видеопамять организована в виде 4 слоев — по одному слою на каждый из основных цветов (красный, зеленый и синий) и еще один слой для признака яркости. Запись в цветовой слой разрешена, если установлен соответствующий бит регистра маскирования памяти. Названия слоев чисто условные (происходят от CGA), поскольку в контроллере VGA четырехразрядный код цвета не подается на монитор непосредственно, а подвергается двукратной перекодировке, превращаясь вначале в 8-разрядный, а затем — в 18-разрядный код.

Регистр также используется в том случае, когда необходимо осуществить загрузку шрифта в знакогенератор видеоконтроллера (в текстовом режиме слой 0 содержит коды символов, слой 1 — их атрибуты, а слой 2 — одну или несколько таблиц растрового представления символов для знакогенератора).

Регистр выбора таблицы символов

Контроллер VGA в текстовых режимах позволяет хранить для знакогенератора в области цветового слоя № 2 до восьми таблиц символов, по 256 символов в каждой. Размер области памяти для одного

символа фиксированный — 32 байта (из которых используются только первые 16), поэтому и размер таблицы тоже строго определенный — 8 Кбайт. Практически всегда применяется только одна таблица — с нулевым номером. Для ее использования в регистр выбора таблицы должен быть записан 0.

Запись информации в регистр выбора таблицы символов допускается только после того, как записан 0 в регистр сброса синхронизатора (то есть выполнен асинхронный сброс).

Регистр режима памяти

Регистр предназначен для управления внутренней организацией памяти видеоконтроллера. Его разряды имеют следующее значение:

- бит 0 — выбор режима (0 — графический, 1 — текстовый);
- бит 1 — объем видеопамяти (0 — 64 Кбайт, 1 — более 64 Кбайт);
- бит 2 — нечетный/четный режим (если установлено значение 0, данные с нечетными адресами отображаются в нечетных слоях памяти, а данные с четными адресами — в четных);
- биты 3–7 — не используются.

Регистры контроллера электронно-лучевой трубки

Эта группа регистров самая большая — она предназначена для задания множества параметров, определяющих свойства видеорежима. Для цветных дисплеев эти регистры адресуются через порты 3D4h и 3D5h.

Регистр адреса КЭЛТ

Регистр адреса КЭЛТ (CRTC Address Register) доступен через порт 3D4h. В него требуется записать номер регистра, с которым после этого можно будет работать через порт 3D5h. Допускается одновременная запись индекса регистра и значения в этот регистр, то есть запись в порт 3D4h можно выполнять 16-разрядными словами. Список регистров КЭЛТ приводится в табл. 4.6. Работать непосредственно можно только с теми регистрами, которые не связаны с формированием основных параметров кадра. Дело в том, что любая ошибка при установке параметров горизонтальной или вертикальной развертки может вывести из строя монитор (может сгореть блок питания или блок развертки монитора, выгореть линия или точка в центре экрана электронно-лучевой трубки и т. д.). Ниже рассматриваются

только те регистры, работа с которыми представляет интерес для программиста и не связана с каким-либо риском.

Таблица 4.6. Регистры контроллера ЭЛТ

Индекс	Регистр
00h	Полная длина линии горизонтальной развертки
01h	Длина отображаемого участка горизонтальной развертки
02h	Начало затемнения по горизонтали
03h	Конец затемнения по горизонтали
04h	Начало обратного хода луча по горизонтали
05h	Конец обратного хода луча по горизонтали
06h	Общее число строк развертки
07h	Дополнение
08h	Предварительная строчная развертка
09h	Высота текстовых символов
0Ah	Начальная линия курсора
0Bh	Конечная линия курсора
0Ch	Старший байт начального адреса
0Dh	Младший байт начального адреса
0Eh	Старший байт адреса курсора
0Fh	Младший байт адреса курсора
10h	Начало обратного хода луча по кадру
11h	Конец обратного хода луча по кадру
12h	Конец видимой части кадра
13h	Логическая ширина экрана
14h	Положение подчеркивания символа
15h	Начало затемнения по кадру
16h	Конец затемнения по кадру
17h	Управление режимом
18h	Сравнение строк

Регистр начальной линии курсора

Регистр начальной линии курсора (Cursor Start Register) имеет индекс 0Ah. Он определяет номер строки знакоместа, с которой начинается вывод курсора (нумерация строк ведется с 0).

Разряды регистра имеют следующее значение:

- биты 0–4 — номер начальной линии курсора;
- бит 5 — гашение курсора (если разряд установлен в 1, то курсор становится невидимым);
- биты 6–7 — не используются.

Регистр конечной линии курсора

Регистр конечной линии курсора (Cursor End Register) имеет индекс 0Bh. Он определяет номер строки знакоместа, на которой завершается отображение курсора.

Разряды регистра имеют следующее значение:

- биты 0–4 — номер конечной линии курсора;
- биты 5 и 6 — отклонение курсора (оба разряда должны быть установлены в 0);
- бит 7 — не используется.

Содержимое регистров начальной и конечной линий курсора имеет значение только для текстовых режимов.

Регистры старшего и младшего байтов начального адреса

Регистры старшего байта начального адреса (Start Address High Register) и младшего байта начального адреса (Start Address Low Register) имеют индексы 0Ch и 0Dh соответственно. Они содержат старший и младший байты 16-разрядного адреса видеопамяти, с которого начинается вывод данных на экран. Изменяя начальный адрес, можно выполнять прокрутку изображения по горизонтали и по вертикали, а также осуществлять переключение видеостраниц. Прокрутка по горизонтали осуществляется увеличением или уменьшением значения начального адреса на 1, прокрутка по вертикали — прибавлением к начальному адресу или вычитанием из него логической длины строки в байтах. Переключение страниц осуществляется путем записи в регистры начального адреса произведения номера страницы на размер страницы в байтах (нумерация страниц ведется с 0).

К сожалению, этими регистрами можно пользоваться только в текстовых режимах и старых графических режимах VGA (в новых режимах 16-разрядного числа недостаточно).

Регистры старшего и младшего байтов адреса курсора

Регистры старшего байта адреса курсора (Cursor Location High Register) и младшего байта адреса курсора (Cursor Location Low Register) имеют индексы 0Eh и 0Fh соответственно. Они содержат старший и младший байты 16-разрядного смещения курсора относительно начала экрана (левого верхнего угла). Это смещение отсчитывается как сумма номера колонки и произведения номера строки на длину строки в символах. Содержимое регистров адреса курсора имеет значение только для текстовых режимов.

Регистр логической ширины экрана

Регистр логической ширины экрана (Offset Register) имеет индекс 13h. Он определяет объем видеопамяти, который выделяется для хранения одной строки изображения. Иными словами, он задает логическую длину строки, которая может быть больше физической (реально отображаемой на экране). Применяется этот регистр для выравнивания длины строки в пикселах на значение, кратное 2^N, что часто необходимо делать с целью ускорения вывода информации на экран (хотя расплачиваться приходится перерасходом видеопамяти).

Регистры графического контроллера

С помощью регистров графического контроллера осуществляется обработка потока данных между процессором и видеопамятью. Применяются эти регистры только в старых 16-цветных графических режимах. Для адресации используется индексный порт 3CEh (регистр адреса графики), передача данных выполняется через порт 3CFh. Список адресуемых регистров приведен в табл. 4.7.

Таблица 4.7. Регистры графического контроллера

Индекс	Регистр
0	Установка/сброс (Set/Reset Register)
1	Разрешение установки/сброса (Enable Set/Reset Register)
2	Сравнение цветов (Color Compare Register)
3	Циклический сдвиг данных (Data Rotate Register)
4	Выбор схемы чтения (Read Map Select Register)

продолжение ⇗

Таблица 4.7 (*продолжение*)

Индекс	Регистр
5	Режимы записи и считывания (Mode Register)
6	Смешанные данные (Miscellaneous Register)
7	Цвет безразличен (Color Don't Care)
8	Битовая маска (Bit Mask Register)

Регистры установки/сброса, разрешения установки/сброса, сравнения цветов, циклического сдвига данных применяются только в 16-цветных графических режимах и далее не рассматриваются. В любых других режимах в эти регистры должны быть записаны нули.

Регистры безразличия цвета и битовой маски также применяются только в 16-цветных режимах. В других режимах они не нужны, но запись в эти регистры неверных значений может искажать результаты операций ввода-вывода. В регистре безразличия цвета должны быть записаны 0 в текстовых режимах и значение 0Fh в графических; в регистр битовой маски нужно записать значение FFh.

Регистр выбора схемы чтения

В 16-цветных графических режимах видеопамять разделена на четыре слоя. Чтобы прочитать информацию из какого-либо слоя (в режиме чтения 0), нужно вначале установить номер этого слоя в данном регистре. Разряды регистра имеют следующее значение:

- биты 0 и 1 — номер считываемой битовой матрицы (от 0 до 3);
- биты 2–7 — не используются.

Регистр режимов записи и считывания

Видеоконтроллер VGA позволяет использовать три метода записи данных и два метода чтения (по умолчанию используются режим чтения 0 и режим записи 0). Выбор оптимального режима для конкретной задачи значительно ускоряет выполнение операций вывода и чтения пикселов. Значение разрядов регистра следующее:

- биты 0 и 1 — выбор режима записи (0 — непосредственная запись, 1 — запись с использованием регистров-защелок, 2 — запись с использованием битовой маски);
- бит 2 — не используется (установлен в 0);
- бит 3 — выбор режима чтения;

- бит 4 — четный/нечетный режим (0 — для графических режимов, 1 — для текстовых);
- бит 5 — 4-цветный режим (устанавливается в 1 в видеорежимах 04h и 05h);
- бит 6 — 256-цветный режим (устанавливается в 1 в режиме 13h);
- бит 7 — не используется (установлен в 0).

Регистр смешанных данных

Регистр получил такое название потому, что разработчики «запихнули» в него информацию, которая не поместились в других регистрах графического контроллера. Разряды регистра смешанных данных имеют следующее назначение:

- бит 0 — выбор режима (0 — текстовый, 1 — графический); в графическом режиме запрещена генерация символов и разрешена адресация по пикселам;
- бит 1 — не используется;
- биты 2 и 3 — положение в адресном пространстве и размер окна для доступа к видеопамяти (0 — адрес A0000h, размер 128 Кбайт; 1 — адрес A0000h, размер 64 Кбайт; 2 — адрес B0000h, размер 32 Кбайт; 3 — адрес BB000h, размер 32 Кбайт);
- биты 4–7 — не используются.

Регистры контроллера атрибутов

Регистры контроллера атрибутов отвечают за перекодировку цветов (4-разрядный код преобразуется в 8-разрядный). Кроме того, один из регистров задает цвет рамки экрана, а другой обеспечивает плавную прокрутку изображения по горизонтали в текстовых и 16-цветных режимах.

Разработчики контроллера атрибутов превзошли всех остальных по части экономии пространства ввода-вывода — поместили по одному и тому же адресу 3C0h индексный порт и порт данных. В результате обмен информацией между центральным процессором компьютера и контроллером атрибутов выполнялся слишком медленно, и в последующие модификации контроллеров VGA было внесено изменение — теперь доступ к регистрам данных допускается также через порт 3C1h (то есть можно использовать механизм одновременной записи в индексный регистр и регистр данных).

Регистр адреса атрибута

Регистр адреса атрибута (Attribute Address Register) доступен для записи через порт 3C0h. Индекс, записанный в регистр адреса атрибута, определяет, какой из регистров данных будет доступен через порт 3C0h при выполнении следующей операции записи или считывания. Значения индексов приведены в табл. 4.8. Разряды регистра распределены следующим образом:

- биты 0–4 — номер адресуемого регистра атрибутов;
- бит 5 — разрешение доступа к регистрам палитры (0 — запретить видеоконтроллеру использование регистров палитры, 1 — разрешить);
- биты 6 и 7 — не используются.

Порядок доступа определяется специальным триггером-счетчиком: при первом обращении передаваемое значение интерпретируется как индекс, при втором — как данные, при третьем — снова как индекс и т. д. Однако перед началом серии операций записи или считывания триггер нужно сбросить. Сброс триггера происходит при считывании данных из первого регистра состояния видеоконтроллера, который в цветных видеорежимах доступен через порт 3DAh.

Запись информации в регистры атрибутов должна производиться только во время обратного хода луча по кадру. Бит 5 индексного регистра при этом должен быть сброшен в ноль. После окончания изменения информации в регистрах атрибутов требуется установить значение бита 5 в единицу путем записи в индексный регистр кода 20h.

Таблица 4.8. Регистры данных контроллера атрибутов

Индекс	Регистр
00h–0Fh	Регистры палитры (Palette Registers)
10h	Управление режимом (Mode Control Register)
11h	Цвет рамки (Overscan Color Register)
12h	Разрешение отображения цветовых слоев (Color Plane Enable Register)
13h	Горизонтальное поэлементное панорамирование (Horizontal PEL Panning Register)
14h	Выбор цвета (Color Select Register)

Регистры палитры

Эти регистры (совместно с регистром выбора цвета) ставят в соответствие 4-разрядным кодам цвета 8-разрядные номера регистров ЦАП, в которых хранятся реальные значения интенсивностей трех компонентов цвета. Назначение разрядов этих регистров следующее:

- биты 0–5 – номер регистра ЦАП (разряды 0–5);
- биты 6 и 7 – не используются.

Регистр управления режимом

Регистр имеет индекс 10h. Он задает режимы работы контроллера атрибутов. Разряды регистра имеют следующее назначение:

- бит 0 – выбор режима (0 – текстовый, 1 – графический);
- бит 1 – тип дисплея (0 – цветной, 1 – монохромный);
- бит 2 – разрешение использования 9-битных символов псевдографики в монохромном текстовом режиме (0 – использование запрещено, 1 – разрешено); в цветных режимах требуется установить этот разряд в 0;
- бит 3 – определяет значение бита 7 атрибута символа в текстовых режимах (0 – мерцание, 1 – яркость фона); в графических режимах этот разряд должен быть установлен в 0;
- бит 4 – не используется (установить в 0);
- бит 5 – управление поэлементным панорамированием верхней области экрана в режиме разделения экрана на 2 части (этот режим нигде не применяется, и значение данного разряда нужно всегда устанавливать в 0);
- бит 6 – управление шагом поэлементного панорамирования (должен иметь значение 1 в 256-цветном режиме и 0 во всех остальных случаях);
- бит 7 – управление замещением битов 4 и 5 регистров палитры (при установке этого разряда в 1 биты 4 и 5 регистров палитры замещаются битами 0 и 1 регистра выбора цвета).

Регистр цвета рамки

Регистр имеет индекс 11h и предназначен для задания цвета рамки экрана. В этот регистр заносится номер одного из регистров ЦАП (от 0 до 255), содержимое которого будет определять цвет обрамления.

Регистр разрешения отображения цветовых слоев

Регистр с номером 12h используется для ограничения доступа к битовым матрицам со стороны контроллера атрибутов. Ограничение доступа нужно было только для старых видеоконтроллеров с объемом памяти 64 Кбайт и менее. При работе с цветными видеорежимами в этот регистр должно быть записано значение 0Fh. Назначение разрядов регистра:

- бит 0 — доступ к слою 0 (0 — запрещен, 1 — разрешен);
- бит 1 — доступ к слою 1 (0 — запрещен, 1 — разрешен);
- бит 2 — доступ к слою 2 (0 — запрещен, 1 — разрешен);
- бит 3 — доступ к слою 3 (0 — запрещен, 1 — разрешен);
- биты 4–7 — не используются (установить в 0).

Регистр горизонтального поэлементного панорамирования

Регистр имеет номер 13h и предназначен для реализации плавной прокрутки изображения по горизонтали в текстовых и 16-цветных графических режимах.

Дело в том, что в этих режимах увеличение или уменьшение на единицу значения начального адреса приводит к смещению изображения сразу на 8 пикселов. Регистр поэлементного панорамирования дополняет регистры начального адреса, обеспечивая перемещение изображения с точностью до пиксела. Запись информации в этот регистр можно производить только в процессе обратного хода луча по кадру. В регистр можно записывать число от 0 до 7 — будет происходить сдвиг изображения влево на соответствующее число пикселов.

Регистр выбора цвета

Регистр с номером 14h расширяет набор оттенков, доступных в 16-цветных режимах, и обеспечивает быструю смену палитры. Регистр добавляет два старших бита к 6-разрядным значениям, хранящимся в регистрах палитры, обеспечивая тем самым доступ ко всему набору регистров ЦАП. Два старших бита регистров палитры (разряды 4 и 5) могут быть также замещены битами 0 и 1 регистра выбора цвета.

Содержимое регистра выбора цвета воздействует сразу на все 16 регистров палитры, обеспечивая установку одного из 16 возможных наборов оттенков (в любой момент времени доступен только один набор из 16 цветов). Назначение разрядов регистра следующее:

- биты 0 и 1 — заменяют для всех регистров палитры биты 4 и 5, если установлен в единицу разряд 7 регистра управления режимом;
- биты 2 и 3 — определяют разряды 6 и 7 восьмиразрядного номера регистра цвета ЦАП;
- биты 4–7 — не используются.

Регистры цифро-аналогового преобразователя

ЦАП обеспечивает преобразование двоичного кода, хранящегося в памяти видеоконтроллера, в аналоговые сигналы для электронно-лучевой трубки. В состав блока ЦАП входит несколько управляющих регистров и 256 регистров данных, каждый из которых определяет один цвет. Регистр данных имеет длину 18 бит, по 6 бит на каждый основной цвет — красный, зеленый и синий. Таким образом, общее количество возможных оттенков составляет 2^{18} (262 144), но одновременно доступны всего 256 оттенков.

Содержимое таблицы регистров ЦАП влияет на вывод изображения только в текстовых режимах, 16-цветных и 256-цветных графических режимах. В режимах DirectDraw (HiColor и TrueColor) информация поступает прямо на цифро-аналоговые преобразователи (в обход таблицы регистров цвета).

В 256-цветных режимах по умолчанию (то есть после установки режима при помощи прерывания BIOS) первые 16 цветов таблицы ЦАП совпадают с набором цветов 16-цветных режимов. Далее следуют 16 оттенков серого цвета. К сожалению, оставшиеся регистры содержат довольно тусклые цвета. Кроме того, порядок размещения цветов, устанавливаемый по умолчанию, неудобен для трехмерной графики. Поэтому в компьютерных играх обычно приходится заносить в таблицу ЦАП специально подобранные значения, необходимые для решения конкретной задачи. Рассмотрим регистры ЦАП.

Регистр состояния ЦАП

Регистр доступен только для считывания через порт 3C7h. По содержимому регистра можно определить состояние ЦАП. Назначение разрядов регистра:

- биты 0 и 1 — код режима доступа к регистрам таблицы ЦАП (0 — регистры данных доступны только для чтения, 3 — только для записи; коды 1 и 2 не определены);
- биты 2–7 — зарезервированы.

Регистр выбора считываемого регистра цветовой таблицы

Регистр доступен только для записи через порт 3C7h. В него заносится номер регистра таблицы цветов, значение из которого требуется прочитать.

После выполнения последовательного считывания из регистра данных значений трех основных цветов индекс в регистре выбора считываемого регистра цветовой таблицы автоматически увеличивается на единицу. Это позволяет прочитать группу регистров данных, не записывая каждый раз новый индекс.

Регистр выбора записываемого регистра цветовой таблицы

Регистр доступен через порт 3C8h. В него записывается номер регистра таблицы цветов, в который нужно занести новое значение. После выполнения последовательной записи в регистр данных значений трех основных цветов индекс в регистре выбора записываемого регистра цветовой таблицы автоматически увеличивается на единицу, что позволяет записать группу регистров данных, не записывая каждый раз новый индекс.

Регистр данных цветовой таблицы ЦАП

Через порт 3C9h для чтения или для записи доступен один из регистров данных цветовой таблицы ЦАП, номер которого записан в соответствующем индексном регистре. Запись и считывание выполняются в три этапа, каждый раз передается 6 разрядов, соответствующих одному из основных цветов: первые 6 бит определяют интенсивность красного цвета, вторые — зеленого, третьи — синего. Нарушать процесс передачи информации нельзя, поэтому нужно запретить прерывания, пока не будет завершена запись или считывание всех трех 6-битовых групп. Назначение разрядов регистра данных следующее:

- биты 0–5 — интенсивность одного из основных цветов;
- биты 6 и 7 — не используются.

Чтение и запись данных в цветовую таблицу ЦАП можно выполнять только в течение обратного хода луча по кадру, что создает определенные проблемы: операции с портами ввода-вывода выполняются очень медленно, и одного интервала обратного хода луча может быть недостаточно для перезаписи всей таблицы. В результате на экране появляются разные неприятные видеоэффекты.

Если необходимо использование широкой палитры оттенков, то гораздо удобнее работать в видеорежимах DirectDraw — поток данных из видеопамяти при этом сразу попадает на цифро-аналоговые преобразователи, обходя таблицы перекодировки цветов. Кроме того, режимы DirectDraw обеспечивают гораздо более точное представление цвета и более гладкие цветовые переходы в программах, использующих трехмерную графику.

Особенности работы в текстовом режиме

При включении IBM-совместимого компьютера видеоконтроллер обычно начинает работу с текстового 16-цветного режима с разрешением 80×25 символов (код режима 03h). В этом режиме для работы с видеопамятью выделено окно размером 32 Кбайт в первом мегабайте адресного пространства процессора (начальный линейный адрес окна B8000h).

Текстовый режим до сих пор широко применяется, так как для решения многих задач он является вполне достаточным, отличается простотой, надежностью и невысокими требованиями к аппаратуре компьютера. Основные операции в текстовом режиме: вывод символов, управление курсором и загрузка шрифта.

Видеопамять в текстовом режиме с точки зрения процессора организована следующим образом: на каждый символ приходится по два байта информации, причем первый байт хранит ASCII-код символа, а второй — цвет символа и цвет фона знакоместа этого символа, как показано на рис. 4.3.

	Столбец 0	Столбец 1	Столбец 2	...	Столбец 79
Строка 0	Символ 0	Символ 1	Символ 2	...	Символ 79
	Код | Цвет	Код | Цвет	Код | Цвет		Код | Цввт
Строка 1	Символ 80	Символ 81	Символ 82	...	Символ 159
	Код | Цвет	Код | Цввт	Код | Цвет		Код | Цвет
Строка 2	Символ 160	Символ 161	Символ 162	...	Символ 239
	Код | Цвет	Код | Цвет	Код | Цвет		Код | Цвет
...	...	...	...	...	...
Строка 24	Символ 1920	Символ 1921	Символ 1922	...	Символ 1999
	Код | Цвет	Код | Цвет	Код | Цввт		Код | Цвет

Рис. 4.3. Отображение видеопамяти на экран в текстовом 16-цветном режиме с разрешением 80×25 символов

Для вывода информации в видеопамять обычно используется дополнительный сегментный регистр данных ES, в который перед началом

записи в видеопамять нужно занести число, равное абсолютному начальному адресу буфера, поделенному на 16.

Чтобы вывести символ в заданное знакоместо, нужно помножить номер строки знакоместа на 160 (длину строки в байтах) и прибавить номер столбца знакоместа, после чего записать результат в индексный регистр. Далее в соответствующий байт заносится с помощью косвенной адресации (относительно сегмента ES и избранного индексного регистра) ASCII-код символа. При необходимости значение индексного регистра инкрементируется, и в следующий байт записывается код цвета символа и фона.

Формат кодирования цвета символа показан на рис. 4.4.

- биты 0–3 — цвет символа;
- биты 4–6 — цвет фона;
- бит 7 — мерцание (0 — обычный текст, 1 — мигающий текст).

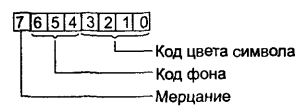


Рис. 4.4. Формат байта описания цвета символа и фона в текстовом режиме

Расшифровка цветовых кодов дана в табл. 4.9 (для фона возможна установка только «темных» цветов с кодами от 0 до 7).

Таблица 4.9. Расшифровка цветовых кодов для текстового режима

Код	Цвет	Код	Цвет
0	Черный	8	Серый
1	Синий	9	Голубой
2	Зеленый	10	Салатный
3	Бирюзовый	11	Светло-бирюзовый
4	Красный	12	Розовый
5	Фиолетовый	13	Светло-фиолетовый
6	Коричневый	14	Желтый
7	Светло-серый	15	Белый

Приведем пример кода программы, в котором осуществляется вывод символа # в 15-й позиции третьей строки экрана, желтым цветом по синему фону.

```
;Загрузить в ES адрес текстовой видеопамяти
        mov     AX,0B800h
        mov     ES,AX
;Умножить номер строки на длину строки в байтах
        mov     AL,160
        mov     AH,3
        mul     AH
;Прибавить к произведению номер колонки
        add     AX,15
;Переписать полученное смещение
;в индексный регистр
        mov     BX,AX
;Записать код символа в видеопамять
        mov     byte ptr ES:[BX],'#'
;Записать код цвета в следующий байт
        inc     BX
        mov     byte ptr ES:[BX],01Eh
```

Кроме вывода символов, в текстовом режиме часто применяются еще два вида операций — перемещение курсора и переключение видеостраниц. Управление курсором осуществляется при помощи регистров видеоконтроллера или функций BIOS. Установить размер прямоугольника текстового курсора можно при помощи регистров начальной и конечной линий курсора или при помощи функции 01h прерывания Int 10h. Перемещение курсора по тексту производится путем записи значения смещения курсора относительно начала видеопамяти в регистры старшего и младшего байта адреса курсора. Позиционировать курсор можно и с помощью функции 02h прерывания Int 10h, но координаты курсора при вызове прерывания задаются в виде номера строки и колонки относительно начала видеостраницы. Если необходимо удалить курсор с экрана, то это можно сделать несколькими способами, наиболее универсальным из которых является перемещение курсора за нижнюю границу экрана (на 25-ю строку).

Переключение видеостраниц осуществляется путем записи смещения левого верхнего угла видеостраницы относительно начала видеопамяти в регистры старшего и младшего байт начального адреса. Эту операцию можно также выполнить при помощи функции 05h прерывания Int 10h. Видеоконтроллер в текстовом режиме обеспечивает 8 видеостраниц, но используется обычно только основная (нулевая) страница. Между тем, дополнительные страницы позволяют выдавать сообщения оператору, не разрушая текст на основной странице. Их можно применять для отображения окна подсказки, сообщений об ошибках, вспомогательных меню, а также для вывода дампа памяти и регистров процессора при отладке программ.

Работа в современных графических режимах

Современные видеоконтроллеры отличаются от VGA-контроллеров тем, что обеспечивают работу с высокими разрешениями и позволяют использовать линейную адресацию видеопамяти. Ниже рассматриваются видеорежимы, реализующие обе указанные возможности.

Организация видеопамяти в 256-цветных режимах

Прототипом этой группы послужил режим VGA с кодом 13h. В 256-цветных режимах каждой точке изображения на экране монитора соответствует один байт видеопамяти, в который записывается код цвета точки. Этот код не используется непосредственно, а служит индексом в специальном массиве, содержащем 256 строк по 3 элемента — таблице цветов ЦАП. Каждый из трех элементов строки таблицы задает интенсивность одного из основных цветов электронно-лучевой трубки (красного, зеленого или синего). Значения интенсивностей, выбранные из строки, соответствующей хранящемуся в видеопамяти коду, поступают в ЦАП.

Видеопамять отображается на экран слева направо и сверху вниз, как показано на рис. 4.5 (обратите внимание: экранная ось Y при этом направлена вниз).

	Столбец 0	Столбец 1	Столбец 2		Столбец 639
Строка 1	Байт 0	Байт 1	Байт 2	...	Байт 639
Строка 2	Байт 640	Байт 641	Байт 642	...	Байт 1279
Строка 3	Байт 1280	Байт 1281	Байт 1282	...	Байт 1283
	...	...	...	...	...
Строка 479	Байт 306560	Байт 306561	Байт 306562	...	Байт 307199

Рис. 4.5. Отображение видеопамяти на экран в 256-цветном режиме с разрешением 640×480 точек

Организация видеопамяти в режимах типа DirectDraw

В режимах группы DirectDraw (HiColor и TrueColor) информация поступает на цифро-аналоговые преобразователи непосредственно из видеопамяти. Соответственно, красная, зеленая и синяя состав-

ляющие цвета точки представлены отдельными полями в выделенной для хранения точки области видеопамяти (от 2 до 4 байт на точку).

В режимах HiColor точка кодируется 16-разрядным словом, причем существует два варианта представления цвета, показанные на рис. 4.6: HiColor15 (формат 1:5:5:5) и HiColor16 (формат 5:6:5). Знаком X на рисунке обозначена зарезервированная (неотображаемая) область данных. Из видеопамяти на экран информация отображается слева направо и сверху вниз.



Рис. 4.6. Форматы данных группы HiColor

В режимах TrueColor для хранения каждого компонента цвета точки выделено по одному байту видеопамяти. Существует два формата представления данных, показанные на рис. 4.7: трехбайтный TrueColor24 и четырехбайтный TrueColor32. Дополнительный байт (на рисунке он обозначен символом X) добавлен в режим TrueColor32 для выравнивания — на экране он не отображается. Дело в том, что процессор может передавать данные видеоконтроллеру только байтами, 16-разрядными и 32-разрядными словами. Чтобы не задеть соседние точки, в 24-битном режиме информацию приходится пересылать по байтам (каждый компонент цвета — отдельно), что в три раза снижает скорость передачи данных. Поскольку такая потеря темпа недопустима во многих приложениях (например, в играх), разработчики решили пожертвовать памятью (в режиме TrueColor32 четверть объема памяти расходуется впустую). Это стандартный прием обмена памяти на производительность, который часто используют инженеры.

Формат TrueColor24

Байт 2	Байт 1	Байт 0
Красный	Зеленый	Синий

Формат TrueColor32

Байт 3	Байт 2	Байт 1	Байт 0
X	Красный	Зеленый	Синий

Рис. 4.7. Форматы данных группы TrueColor

Следует отметить, что на рис. 4.7 расположение информационных разрядов показано в порядке, принятом в литературе по програм-

мированию: старший разряд слева, младший справа. Однако при отображении памяти на экран, как уже было указано, традиционно принят порядок слева направо и сверху вниз. Например, в режиме TrueColor32 1024×768 память отображается на экран в порядке, показанном на рис. 4.8.

	Столбец 0				Столбец 1				...	Столбец 1023			
Строка 0	Точка 0				Точка 1				...	Точка 1023			
	С	З	К	Х	С	З	К	Х		С	З	К	Х
Строка 1	Точка 1024				Точка 1025				...	Точка 2047			
	С	З	К	Х	С	З	К	Х		С	З	К	Х
...	...				...				...	...			
Строка 767	Точка 785408				Точка 785409				...	Точка 786431			
	С	З	К	Х	С	З	К	Х		С	З	К	Х

Рис. 4.8. Отображение видеопамяти на экран в режиме TrueColor32 с разрешением 1024×768 точек

В общем, можно сказать, что развитие видеоконтроллеров направлено в сторону максимального упрощения (для программистов) работы с видеопамятью, даже за счет значительного ее перерасхода (рис. 4.9).

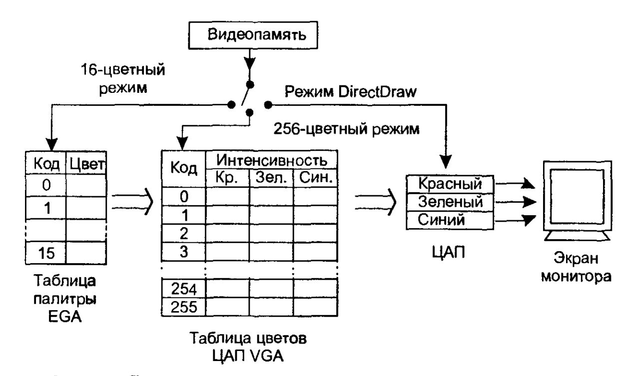


Рис. 4.9. Прохождение информации о цвете пиксела от видеопамяти до экрана монитора в различных графических режимах

Режимы DirectDraw реализуют прямое кодирование цвета без всяких промежуточных преобразований, а последний вариант (TrueColor) обеспечивает простой доступ к каждому из трех основных

цветовых компонентов. Такая организация видеопамяти минимизирует вычисления, ускоряя расчет освещенности объектов при использовании трехмерной графики, но может заметно снижать скорость вывода двухмерного изображения (текста, графика, схемы, чертежа) из-за резкого увеличения объема передаваемой информации.

Режимы адресации и распределение видеопамяти

Как было указано выше, существует два основных метода адресации видеопамяти в графических режимах: устаревший сегментный и современный линейный. В режиме сегментации видеопамять поделена на 64-килобайтные кусочки — окна, причем в каждый момент времени для работы доступен только один такой сегмент. Доступ ко всем сегментам осуществляется через 64-килобайтный участок адресного пространства процессоров x86 с абсолютным адресом A0000h. Режим сегментации имеет два серьезных недостатка, заметно снижающих скорость работы видеосистемы:

- при выводе изображения необходимо постоянно контролировать пересечение границ сегментов, что требует выполнения процессором ряда дополнительных команд;
- при пересечении границ сегментов приходится выполнять их переключение посредством программных процедур VESA BIOS, поскольку механизм переключения при помощи регистров видеоконтроллера так и не был стандартизирован.

Указанные недостатки режима сегментации практически незаметны при выводе статических изображений (рисунков, фотографий, карт, чертежей), однако весьма заметно проявляют себя при использовании анимации.

Линейная адресация обеспечивает гораздо более простой и удобный режим работы — вся видеопамять представляет собой с точки зрения процессора единый (непрерывный) участок адресного пространства. В линейном режиме нет никаких искусственных границ и, соответственно, не нужно тратить время на контроль их пересечения. Основным недостатком метода линейной адресации является невозможность его использования в реальном режиме DOS и режиме виртуальных процессоров. Линейная адресация доступна только в защищенном режиме и режиме линейной адресации данных.

Видеорежим и способ адресации видеопамяти задаются одновременно — кодом номера режима при вызове функции переключения

видеорежимов VESA BIOS (прерывание Int 10h, функция 4Fh, подфункция 02h). Прежде чем устанавливать режим, желательно убедиться, что видеоконтроллер его поддерживает, для чего нужно использовать подфункцию 01h функции 4Fh прерывания Int 10h (см. листинг 4.1).

Распределением видеопамяти управляет программист. При работе со статическим изображением обычно используется только первая видеостраница, а организация памяти сохраняется в том виде, который устанавливается по умолчанию при включении видеорежима (логическая длина видеостроки в пикселах равна физическому разрешению экрана).

При работе с анимацией часто используют две видеостраницы, а память реорганизуют с целью максимального ускорения вывода изображения. Пока процессор перерисовывает изображение в одной области памяти, видеоконтроллер осуществляет вывод предыдущего кадра на экран из другой области (рис. 4.10). Для реорганизации памяти применяются подфункции 06h и 07h функции 4Fh прерывания Int 10h. Подфункция 06h позволяет сделать логическую длину строки больше физической и используется для выравнивания длины строки на 2^N. Выравнивание позволяет ускорить вычисление координат, упростить контроль границ и создать невидимую вертикальную защитную полосу. Подфункция 07h позволяет выбирать положение страниц в видеопамяти и переключать страницы.

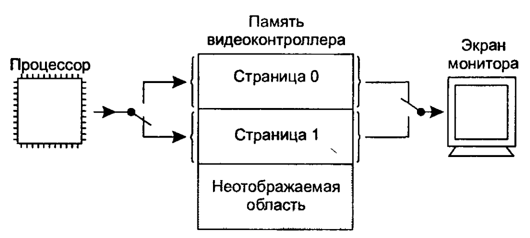


Рис. 4.10. Распределение видеопамяти при использовании механизма переключения страниц

Рисование линий при линейной адресации памяти

Сегментация видеопамяти существенно замедляет рисование линий в режимах с высоким разрешением, особенно при использовании

для переключения страниц прерываний VESA BIOS: при рисовании прямых под большим углом к горизонтали переключение приходится осуществлять через каждые несколько десятков точек. Переключение через регистры видеоконтроллера могут выполнять только Windows-драйверы, написанные программистами фирмы-изготовителя, поскольку документация контроллера доступна только для них. Драйверы для других операционных систем обычно не предоставляются, и, следовательно, вывод линий осуществляется в режиме сегментации очень медленно.

Линейная адресация позволяет избежать переключения страниц. Модель памяти, как было показано в предыдущих разделах, при линейной адресации очень простая: видеоконтроллер воспринимает свою оперативную память как прямоугольный массив (N строк, M столбцов), а процессор видит память контроллера как линейный массив размером N×M×K байт, где K — число байтов на точку изображения.

Рисование линии базируется на операции вывода точки. Вывод точки в современных видеорежимах заключается в том, что процессор записывает по соответствующему адресу видеопамяти код цвета точки, например, при помощи команды MOV. При использовании линейной адресации для обращения к памяти можно применять в качестве индексного любой из регистров общего назначения, а в качестве сегментного — регистр данных, настроенный на видеопамять (в защищенном режиме) или на линейную адресацию. Например, если для включения линейной адресации применялась процедура, описанная ранее в разделе «Функции VESA BIOS», вывод точки белого цвета в 256-цветном режиме может быть осуществлен следующей командой:

```
mov     [byte ptr GS:EDI],0Fh
```

Здесь 0Fh — код белого цвета, а регистр EDI — абсолютный адрес точки, то есть сумма начального адреса видеопамяти и смещения точки от начала видеопамяти.

В режиме HiColor при выполнении такой же операции нужно записать не байт, а 16-разрядное слово:

```
mov     [word ptr GS:EDI],0FFFFh
```

В режиме TrueColor32 вывод точки белого цвета заключается в записи соответствующего 32-разрядного кода:

```
mov     [word ptr GS:EDI],0FFFFFFh
```

Процесс рисования горизонтальной линии заключается в циклическом выводе кода цвета линии в ячейки памяти, соответствующие

соседним точкам (после вывода каждой точки к значению индексного регистра прибавляется количество байтов на точку). Для 256-цветного режима указанный процесс выглядит так:

```
; Установить цвет линии
        mov     AL,[LineColor]
; Записать адрес начальной точки в индексный регистр
        mov     EDI,[StartPixelAddress]
; Задать длину линии в пикселах
        mov     CX,[LineLength]
@@L:    mov     [GS:EDI],AL ;нарисовать пиксел
        inc     EDI
        loop    @@L
```

В режиме TrueColor32 та же операция выполняется следующим образом:

```
; Установить цвет линии
        mov     EAX, [LineColor]
; Записать адрес начальной точки в индексный регистр
        mov     EDI,[StartPixelAddress]
; Задать длину линии в пикселах
        mov     CX,[LineLength]
@@L:    mov     [GS:EDI],EAX ;нарисовать пиксел
        add     EDI,4
        loop    @@L
```

Процесс рисования вертикальной линии отличается от описанного выше тем, что для перехода на одну точку вниз к содержимому индексного регистра прибавляется длина логической строки в байтах (которая вычисляется как произведение длины строки в пикселах на размер пиксела в байтах). Например, для 256-цветного режима цикл рисования выглядит так:

```
; Установить цвет линии
        mov     AL,[LineColor]
; Записать адрес начальной точки в индексный регистр
        mov     EDI,[StartPixelAddress]
; Задать длину линии в пикселах
        mov     CX,[LineLength]
@@L:    mov     [GS:EDI],AL  ;нарисовать пиксел
        ; Прибавить длину строки в байтах
        add     EDI,LogicalStringLength
        loop    @@L
```

Так же легко реализуется алгоритм рисования наклонных линий, имеющих постоянный размер шага, то есть состоящих из отрезков одинаковой длины. Например, при выводе диагональной линии под углом 45 градусов одновременно выполняется перемещение на одну точку по горизонтали и по вертикали.

В режиме HiColor для горизонтальной линии получается следующий цикл:

```
; Установить цвет линии
        mov     AX,[LineColor]
; Записать адрес начальной точки в индексный регистр
        mov     EDI,[StartPixelAddress]
; Задать длину линии в пикселах
        mov     CX,[LineLength]
@@L:    mov     [GS:EDI],AX
        ; Прибавить сумму длины строки
        ; и размера пиксела в байтах
        add     EDI,(LogicalStringLength+1)*2
        loop    @@L
```

Для вывода произвольных линий можно использовать целочисленный алгоритм Брезенхема, описанный во всех книгах по компьютерной графике. Наиболее простое описание алгоритмов рисования линий, окружностей и эллипсов приведено в книге [1], но перевод с языка С на ассемблер теперь выполняется гораздо проще, чем десять лет назад, так как уже не нужно манипулировать с масками точек (основная «головная боль» в 16-цветных режимах), а можно использовать 32-разрядные регистры процессора и линейную адресацию видеопамяти. Полностью листинги примеров рисования линий всех типов, в том числе по алгоритму Брезенхема, будут показаны ниже — вместе с выводом текста.

Вывод текста и статических изображений в графических режимах

Функции BIOS VGA не обеспечивают приемлемой скорости вывода символов на экран, поэтому в графических режимах текст выводится напрямую в видеопамять (растровое разложение символа передается по точкам).

Наиболее простым способом вывода на экран текста и статических изображений является метод масок. Вообще говоря, для создания изображения объекта необходимо два массива: массив, содержащий рисунок объекта, и массив, содержащий маску прозрачности, то есть указание, какие точки рисунка должны отображаться на экране, а в каких надлежит сохранить фон изображения. Однако с целью экономии памяти оба массива объединяют и именуют полученный результат маской объекта (рис. 4.11). Таким образом, маска представляет собой растровое изображение объекта, совмещенное с шаблоном прозрачности.

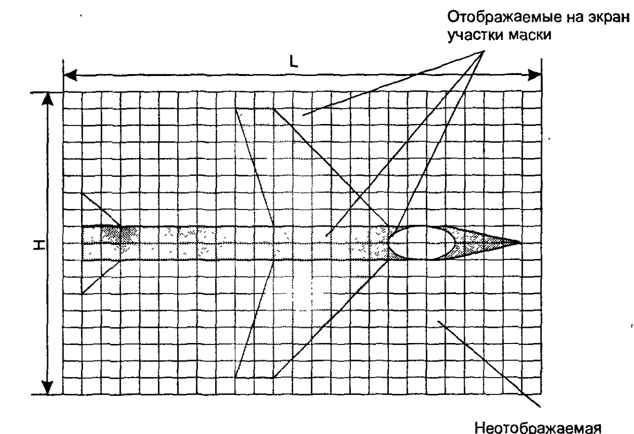


Рис. 4.11. Маска объекта «самолет»

Дело в том, что изображение объекта обычно хранится в виде матрицы, то есть имеет прямоугольную форму, но форма самого объекта редко является прямоугольной. Шаблон прозрачности показывает, какие части растрового изображения относятся к объекту и должны быть отображены на экране, а какие — не относятся и не отображаются. При совмещении шаблона прозрачности с изображением объекта в качестве признака прозрачности выбирают один из цветовых кодов, чаще всего — ноль.

Вывод изображения объекта по методу маски в простейшем случае заключается в копировании маски из оперативной памяти в соответствующий этому изображению участок видеопамяти. Те точки маски, значение кода которых соответствует признаку прозрачности, не копируются, а пропускаются — на их месте в видеопамяти будут сохраняться точки фона.

При выводе текста (или небольших статических изображений) часто бывает целесообразно применять разметку экрана на знакоместа (как в текстовом режиме), так как это позволяет упростить алгоритмы вывода на экран и ускорить выполнение соответствующий операций. Полный размер знакоместа может быть больше размера маски символа, если между символами вставить пустые строки

и колонки — читабельность текста в результате несколько улучшается, но снижается емкость экрана в знакоместах (рис. 4.12). Однако обычно символы выводятся вплотную друг к другу, без дополнительных промежутков.

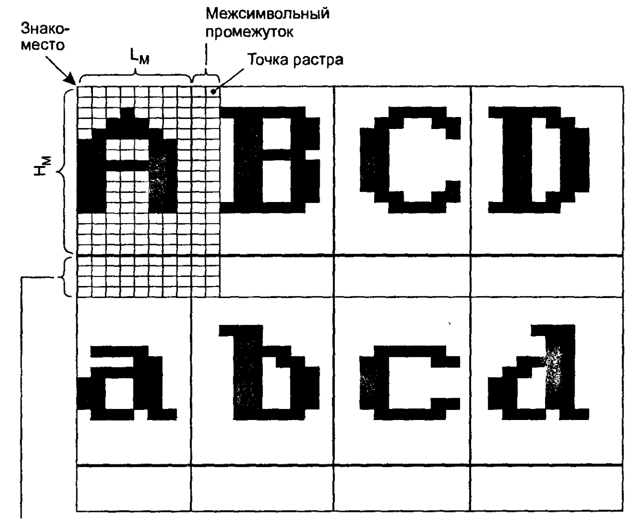


Рис. 4.12. Разметка экрана на знакоместа при выводе текста в графическом режиме ($L_м$ — ширина маски символа, $H_м$ — высота маски)

Если необходимо отображать одинаковые по форме объекты различного цвета, например, выводить текстовые символы различными цветами, то маска применяется для кодирования не цвета, а яркости точек изображения. В этом случае процессор не копирует маску прямо в видеопамять, а выполняет вначале сложение базового кода цвета со значением яркости (в 256-цветных режимах) или распределение кода яркости по заданным цветовым компонентам RGB (в режимах DirectDraw).

Рассмотрим теперь примеры выполнения простых операций в различных графических режимах. В листинге 4.2 собраны универсальные

процедуры, предназначенные для установки графических VESA-режимов, вывода текстовой и числовой информации в видеопамять.

- Процедура GrabRusFont считывает текущий шрифт 8×16 прямо из памяти видеоконтроллера (по методике, предложенной в [29]) и записывает его в массив Font8x16. Перед запуском подпрограммы должен быть установлен текстовый режим, а сразу после ее выполнения желательно выполнить переустановку видеорежима.
- Процедура SetVESAVideoMode проверяет наличие режима VESA с номером, заданным константой GraphicsMode, запоминает в глобальной переменной LinearVideoBuffer начальный абсолютный (линейный) адрес памяти видеоконтроллера, устанавливает видеорежим, а затем увеличивает логическую длину строки экрана до 1024 пикселов. В случае если режим не поддерживается, процедура завершается немедленным (аварийным) выходом из программы.
- Процедура SetTrueColor32 осуществляет поиск видеорежима TrueColor32 с разрешением, заданным константами ScreenLength и ScreenHeigth, устанавливает его (если он поддерживается видеоконтроллером) и задает логическую длину строки экрана 1024 пиксела. В случае если подходящий режим не обнаружен, процедура завершается немедленным выходом из программы.
- Процедура WaitVSync предназначена для синхронизации вывода динамических изображений — она выполняет операцию ожидания начала обратного хода луча по кадру.
- Процедура GShowString выводит строку текста в заданную область экрана, используя цвет символов и фона, заданный по умолчанию.
- Процедура GShowByteHexCode отображает в заданную позицию экрана содержимое регистра AL (байт данных) в шестнадцатеричном коде.
- Процедура GShowHexWord отображает в заданную позицию экрана содержимое регистра AX (слово данных) в шестнадцатеричном коде.
- Процедура GShowHexDWord отображает в заданную позицию экрана содержимое регистра EAX (двойное слово) в шестнадцатеричном коде.
- Процедура GShowByteBinCode отображает в заданную позицию экрана содержимое регистра AL (байт данных) в двоичном коде.

- Процедура GShowBinDWord отображает в заданную позицию экрана содержимое регистра EAX (двойное слово) в двоичном коде.
- Процедура ShowColorString выводит текстовую строку заданного цвета в заданную область экрана.

Листинг 4.2. Процедуры общего назначения, предназначенные для установки графических режимов и работы в них

```
DATASEG
; Место для хранения информации VESA BIOS
VESA_BIOS  DB 512 DUP(?)
; Область для приема информации о параметрах видеорежима
VESA_info  DB 256 DUP(?)
; Физический адрес линейного видеобуфера
LinearVideoBuffer DD ?
; Буфер для сохранения шрифта (16x256 байт)
Font8x16   DB 4096 DUP(?)
; Позиция отображаемого символа
FontString DW ? ;номер строки шрифта
FontColumn DW ? ;номер колонки шрифта
ErrMod DB 12,17,"Заданный режим не "
       DB "поддерживаются контроллером",0
ErrPrm DB 12,17
       DB "Некорректный параметр у функции GShowString",0
ENDS

CODESEG
;****************************************************
;*      СЧИТЫВАНИЕ "РУССКОГО" ШРИФТА ИЗ ПАМЯТИ      *
;*                  ВИДЕОКОНТРОЛЛЕРА                *
;*         (процедура параметров не имеет)          *
;* Операция захвата шрифта требует перепрограммиро- *
;* вания регистров видеоконтроллера и ее следует    *
;* выполнять непосредственно перед переходом из     *
;* текстового режима в графический, чтобы не восста-*
;* навливать регистры (при смене режима они все     *
;* равно будут перезаписаны).                       *
;****************************************************
PROC GrabRusFont near
        pushad
; Перепрограммировать синхронизатор
        cli
        mov     DX,3C4h
        ; Установить последовательную адресацию
        ; ячеек видеопамяти
        mov     AX,0704h
        out     DX,AX
        sti
; Перепрограммировать графический контроллер
```

продолжение ⇨

Листинг 4.2 (*продолжение*)

```
        mov     DX,3CEh
        ; Выбрать для считывания плоскость 2
        mov     AX,0204h
        out     DX,AX
        ; Запретить четную-нечетную адресацию
        mov     AX,0005h
        out     DX,AX
        ; Установить окно доступа по адресу A0000h
        mov     AX,0006h
        out     DX,AX

; Скопировать шрифт в буфер FontBx16
        mov     AX,0A000h
        mov     ES,AX
        mov     SI,0
        mov     BX,offset FontBx16
        mov     DX,256
@@M0:   mov     CX,16
@@M1:   mov     AL,[ES:SI]
        mov     [BX],AL
        inc     BX
        inc     SI
        loop    @@M1
        add     SI,16
        dec     DX
        jnz     @@M0
        popad
        ret
ENDP GrabRusFont

;*******************************************
;* УСТАНОВИТЬ ВИДЕОРЕЖИМ ЧЕРЕЗ ФУНКЦИЮ VESA *
;*     (процедура параметров не имеет)      *
;*******************************************
PROC SetVESAVideoMode near
        pushad
        push    ES
        mov     AX,DS
        mov     ES,AX
; Получить информацию о наличии VESA BIOS
        mov     AX,4F00h
        mov     DI,offset VESA_BIOS
        int     10h
; Получить результат выполнения функции
        cmp     AX,4Fh
        jne     @@Err   ;функция не выполнена
        cmp     [dword ptr VESA_BIOS],'ASEV'
        jne     @@Err   ;нет сигнатуры "VESA"
        cmp     [word ptr VESA_BIOS+4],200h
```

```
        jb      @@Err   ;версия "VESA" слишком старая
; Получить параметры видеорежима, номер
; которого задан переменной GraphicsMode
        mov     AX,4F01h
        mov     CX,GraphicsMode
        mov     DI,offset VESA_info
        int     10h
; Выделить адрес линейной области видеопамяти
        mov     EAX,[offset VESA_info+28h]
        mov     [LinearVideoBuffer],EAX
; Установить заданный видеорежим
        mov     BX,GraphicsMode
        mov     AX,4F02h
        int     10h
; Устанавливаем заданную логическую ширину строки
        xor     BX,BX
        mov     CX,LogicalStringLength
        mov     AX,4F06h
        int     10h
        pop     ES
        popad
        ret
; Аварийный выход - нет поддержки VESA
@@Err:  MFatalError ErrMod
ENDP SetVESAVideoMode

;*************************************
;* УСТАНОВИТЬ ВИДЕОРЕЖИМ TRUECOLOR32 *
;*  (процедура параметров не имеет)  *
;*************************************
PROC SetTrueColor32 near
        pushad
        push    ES
; Получить информацию о наличии VESA BIOS
        mov     AX,DS
        mov     ES,AX
        mov     AX,4F00h
        mov     DI,offset VESA_BIOS
        int     10h
; Проверить результат выполнения функции
        cmp     AX,4Fh
        jne     @@Err   ;функция не выполнена
        cmp     [dword ptr VESA_BIOS],'ASEV'
        jne     @@Err   ;нет сигнатуры "VESA"
        cmp     [word ptr VESA_BIOS+4],200h
        jb      @@Err   ;версия "VESA" слишком старая
; Найти видеорежим с параметрами TrueColor32 640x480
        ; Загрузить адрес массива номеров
        mov     BX,[word ptr VESA_BIOS+0Eh]
        mov     AX,[word ptr VESA_BIOS+10h]
```

продолжение

Листинг 4.2 (*продолжение*)

```
        mov     ES,AX
@@NextMode:
        ; Получить параметры очередного видеорежима
        mov     CX,[ES:BX] ;загрузить номер режима
        cmp     CX,0       ;ошибка?
        je      @@Err
        cmp     CX,0FFFFh  ;конец списка?
        je      @@Err
        push    ES
        mov     AX,DS
        mov     ES,AX
        mov     AX,4F01h
        mov     DI,offset VESA_info
        int     10h
        pop     ES
; Проверить значения полей
        ; Горизонтальное разрешение
        cmp     [word ptr VESA_info+12h],ScreenLength
        jne     @@NotTrueColor32
        ; Вертикальное разрешение
        cmp     [word ptr VESA_info+14h],ScreenHeigth
        jne     @@NotTrueColor32
        ; Число битов на пиксел
        cmp     [byte ptr VESA_info+19h],32
        jne     @@NotTrueColor32
        ; Линейный адрес буфера кадра
        cmp     [dword ptr VESA_info+28h],0
        jne     @@SetMode
@@NotTrueColor32:
        inc     BX
        jmp     @@NextMode

@@SetMode:
; Выделить адрес линейной области видеопамяти
        mov     EAX,[offset VESA_info+28h]
        mov     [LinearVideoBuffer],EAX
; Установить найденный режим
        mov     AX,4F02h
        ; Загрузить номер режима
        mov     BX,[ES:BX]
        ; Использовать линейную адресацию
        or      BX,4000h
        int     10h
; Установить логическую ширину строки
        mov     AX,4F06h
        xor     BX,BX
        mov     CX,LogicalStringLength
        int     10h
        pop     ES
```

```
        popad
        ret
; Аварийный выход - нет поддержки заданного режима
@@Err:  MFatalError ErrMod
ENDP SetTrueColor32

;****************************************
;* ОЖИДАНИЕ ОБРАТНОГО ХОДА ЛУЧА ПО КАДРУ *
;****************************************
PROC WaitVSync NEAR
        push  AX
        push  DX
        mov   DX,03DAh ;регистр статуса VGA
@@WaitNotVSyncLoop:
        in    AL,DX
        and   AL,08h ;выделяем бит вертикальной синхронизации
        jnz   @@WaitNotVSyncLoop
@@WaitVSyncLoop:
        in    AL,DX
        and   AL,08h ;выделяем бит вертикальной синхронизации
        jz    @@WaitVSyncLoop
        pop   DX
        pop   AX
        ret
ENDP WaitVSync

;****************************************************
;*          ВЫВОД ТЕКСТОВОЙ СТРОКИ НА ЭКРАН          *
;* Все параметры передаются через одну структуру:    *
;* первый байт - номер начальной строки (0-47);      *
;* второй байт - номер начальной колонки  (0-127);   *
;* далее идет строка, ограниченная нулем.            *
;* Адрес структуры передается через регистры DS:SI.  *
;****************************************************
PROC GShowString near
        push  AX
        push  DX
        cld
; Вычисляем адрес для строки в видеопамяти
        ;загрузить номер строки экрана в DH
        lodsb
        cmp     AL,47
        ja      @@Err ;выход за нижнюю границу экрана
        mov     DH,AL
        ;загрузить номер строки экрана в DL
        lodsb
        cmp     AL,127
        ja      @@Err ;выход за правую границу экрана
        mov     DL,AL
@@L1:   ; Загрузить очередной символ строки в AL
```

продолжение ⇒

Листинг 4.2 (*продолжение*)

```
        lodsb
        ; Проверка на 0 (на конец строки)
        and     AL,AL
        jz      @@L2
        ; Вывести символ на экран
        call    PutGraChar
        jmp     @@L1
; Нормальное завершение
@@L2:   pop     DX
        pop     AX
        ret
; Немедленный выход в DOS при ошибке
@@Err:  MFatalError ErrPrm
ENDP GShowString

;*****************************************************
;*              ВЫВОД ТЕКСТА НА ЭКРАН                *
;* GShowText использует процедуру GShowString для    *
;* вывода на экран группы строк.                     *
;* Параметры:                                        *
;* CX - количество строк;                            *
;* DS:SI - адрес первой строки в группе.             *
;* Строки должны иметь заданный для GShowString формат *
;* и располагаться в памяти последовательно.         *
;* При выводе текста используются принятые по        *
;* умолчанию цвет и фон.                             *
;*****************************************************
PROC GShowText near
; Цикл вывода строк
@@NextString:
        call    GShowString
        loop    @@NextString
; Процедура не сохраняет значения в CX и SI
        ret
ENDP GShowText

;*****************************************************
;*   ВЫВОД БАЙТА НА ЭКРАН В ШЕСТНАДЦАТЕРИЧНОМ КОДЕ   *
;* Подпрограмма выводит содержимое регистра AL       *
;* в шестнадцатеричном коде в указанную позицию      *
;* экрана.                                           *
;* Координаты позиции передаются через глобальные    *
;* переменные ScreenString и ScreenColumn. После     *
;* выполнения операции вывода байта происходит       *
;* автоматическое приращение значений этих переменных. *
;*****************************************************
PROC GShowHexByte near
        pusha
        mov     DH,[byte ptr ScreenString]
```

```
        mov     DL,[byte ptr ScreenColumn]
; Вывести старший разряд числа
        push    AX
        ; Выделить старший разряд
        shr     AL,4
        ; Преобразовать старший разряд в код ASCII
        add     AL,'0'
        cmp     AL,'9'
        jbe     @@M0
        add     AL,'A'-'9'- 1
        ; Вывести разряд числа на экран
@@M0:   call    PutGraChar
        pop     AX
; Вывести младший разряд числа
        ; Выделить младший разряд числа
        and     AL,0FH
        ; Преобразовать младший разряд в код ASCII
        add     AL,'0'
        cmp     AL,'9'
        jbe     @@M1
        add     AL,'A'-'9'- 1
        ; Вывести разряд числа на экран
@@M1:   call    PutGraChar

; Подготовка для вывода следующих байтов
        ; Перевести текущую позицию на 2 символа влево
        add     [ScreenColumn],2
        ; Проверить пересечение правой границы экрана
        cmp     [ScreenColumn],80
        jb      @@End
        ; Если достигнута правая граница экрана - перейти
        ; на следующую строку
        sub     [ScreenColumn],80
        inc     [ScreenString]
; Конец подпрограммы
@@End:  popa
        ret
ENDP GShowHexByte

;****************************************************
;*        ВЫВОД 16-РАЗРЯДНОГО СЛОВА НА ЭКРАН        *
;*              В ШЕСТНАДЦАТЕРИЧНОМ КОДЕ            *
;* Параметры:                                       *
;* AX - число, которое будет выведено на экран.     *
;* Номер строки передается через глобальную         *
;* переменную ScreenString, номер столбца - через   *
;* переменную ScreenColumn, цвет текста определяется *
;* переменными DefaultColor и DefaultBackground.     *
;****************************************************
PROC GShowHexWord NEAR
```

продолжение ⇒

Листинг 4.2 (*продолжение*)

```
        xchg    AL,AH
        call    GShowHexByte
        xchg    AL,AH
        call    GShowHexByte
        ret
ENDP GShowHexWord

;*************************************************
;*        ВЫВОД 32-РАЗРЯДНОГО СЛОВА НА ЭКРАН      *
;*              В ШЕСТНАДЦАТЕРИЧНОМ КОДЕ          *
;* Параметры:                                     *
;* EAX - число, которое будет выведено на экран.  *
;* Номер строки передается через глобальную       *
;* переменную ScreenString, номер столбца - через *
;* переменную ScreenColumn, цвет текста определяется *
;* переменными DefaultColor и DefaultBackground.  *
;*************************************************
PROC GShowHexDWord NEAR
        rol     EAX,8
        call    GShowHexByte
        rol     EAX,8
        call    GShowHexByte
        rol     EAX,8
        call    GShowHexByte
        rol     EAX,8
        call    GShowHexByte
        ret
ENDP GShowHexDWord

;****************************************************
;*        ВЫВОД БАЙТА НА ЭКРАН В ДВОИЧНОМ КОДЕ       *
;* Подпрограмма выводит содержимое регистра AL       *
;* в двоичном коде в указанную позицию экрана.       *
;* Координаты позиции передаются через глобальные    *
;* переменные ScreenString и ScreenColumn. После     *
;* выполнения операции вывода байта происходит       *
;* автоматическое приращение значений этих переменных. *
;****************************************************
PROC GShowBinByte near
        pusha
        mov     DH,[byte ptr ScreenString]
        mov     DL,[byte ptr ScreenColumn]
        ; Копируем отображаемый байт в BL
        mov     BL,AL
; Отобразить разряды числа (начиная со старшего)
        mov     CX,8   ;счетчик разрядов
@@L0:   mov     AL,'0'
        ; Выделить очередной разряд числа
        rol     BL,1
```

```
        jnc     @@L1
        mov     AL,'1'
        ; Вывести разряд числа на экран
@@L1:   call    PutGraChar
        loop    @@L0
; Подготовка для вывода следующих байтов
        ; Перевести текущую позицию на B символов влево
        add     [ScreenColumn],B
        ; Проверить пересечение правой границы экрана
        cmp     [ScreenColumn],80
        jb      @@End
        ; Если достигнута правая граница экрана -
        ; перейти на следующую строку
        sub     [ScreenColumn],B0
        inc     [ScreenString]
; Конец подпрограммы
@@End:  popa
        ret
ENDP GShowBinByte

;*****************************************************
;* ВЫВОД 32-РАЗРЯДНОГО СЛОВА НА ЭКРАН В ДВОИЧНОМ КОДЕ *
;* Параметры:                                        *
;* EAX - число, которое будет выведено на экран.     *
;* Номер строки передается через глобальную          *
;* переменную ScreenString, номер столбца - через    *
;* переменную ScreenColumn, цвет текста определяется *
;* переменными DefaultColor и DefaultBackground.      *
;*****************************************************
PROC GShowBinDWord NEAR
        rol     EAX,B
        call    GShowBinByte
        inc     [ScreenColumn]
        rol     EAX,B
        call    GShowBinByte
        inc     [ScreenColumn]
        rol     EAX,8
        call    GShowBinByte
        inc     [ScreenColumn]
        rol     EAX,8
        call    GShowBinByte
        ret
ENDP GShowBinDWord
ENDS
```

Универсальная процедура вывода текстовой строки опирается на процедуру вывода символа, *специфическую для каждого типа видеорежима.* В листинге 4.3 показан вариант реализации процедуры вывода символа PutGraChar и процедуры очистки экрана GClearScreen для 256-цветных режимов с линейным видеобуфером.

ВНИМАНИЕ

Стандартные растровые шрифты MS-DOS с размером маски символа 8x16 целесообразно использовать только при низком разрешении экрана (до 800x600 включительно). При разрешении 1024x768 и выше программист вынужден создавать (при помощи типовой или самодельной программы — генератора шрифтов) свой собственный шрифт, увеличенный в 1,5–2 раза (12x24 или 16x32).

Листинг 4.3. Процедуры вывода символа и очистки экрана для 256-цветных режимов с линейной адресацией видеобуфера

```
DATASEG
; Цвет текста в графическом режиме по умолчанию
DefaultColor      DB WHITE  ;белый
; Цвет фона в графическом режиме по умолчанию
DefaultBackground DB BLACK  ;черный
ENDS

CODESEG
;****************************************************
;* ВЫВОД СИМВОЛА 8x16 НА ЭКРАН В ГРАФИЧЕСКОМ РЕЖИМЕ *
;*             (для 256-цветных режимов)            *
;* Все параметры передаются через регистры:         *
;* AL - ASCII-код символа;                          *
;* DH - номер текстовой строки экрана;              *
;* DL - номер текстовой колонки экрана;             *
;* Используются цвет символов и цвет фона,          *
;* заданные по умолчанию.                           *
;****************************************************
PROC PutGraChar NEAR
        push    DS
        pushad
        mov     CX,[CS:MainDataSeg]
        mov     DS,CX
        cld
        ; Смещение символа от начала шрифта
        mov     SI,offset Font8x16
        xor     AH,AH
        shl     AX,4
        add     SI,AX
        ; Вычислить левый верхний угол символа
        xor     EBX,EBX
        mov     BL,DH
        shl     EBX,14 ;умножить номер строки на 16*1024
        xor     DH,DH
        shl     DX,3   ;умножить номер столбца на 8
        or      BX,DX
        mov     EDI,EBX
        add     EDI,[LinearVideoBuffer]
```

```
        mov     BL,[DefaultColor]
        mov     DL,[DefaultBackground]

        mov     AH,16 ;счетчик строк иаски буквы
@@M0:   lodsb
        mov     CX,8  ;счетчик точек в строке иаски
@@M1:   rol     AL,1
        jc      @@M2
        mov     [GS:EDI],DL
        jmp     @@M3
@@M2:   mov     [GS:EDI],BL
@@M3:   inc     EDI
        loop    @@M1
        add     EDI,LogicalStringLength-8
        dec     AH
        jnz     @@M0
; Завершение процедуры
@@EndPutGraChar:
        popad
        pop     DS
        inc     DL
        ret
ENDP PutGraChar

;**************************************
;* ОЧИСТКА ЭКРАНА В ГРАФИЧЕСКОМ РЕЖИМЕ *
;*   (процедура параметров не имеет)   *
;**************************************
PROC GClearScreen NEAR
        pushad
; Умножить высоту экрана ScreenHeigth на логическую
; ширину строки 1024 пиксела
        mov     ECX,ScreenHeigth
        shl     ECX,10
; Загрузить в индексный регистр линейный
; адрес видеопамяти
        mov     EDI,[LinearVideoBuffer]
; Заполнить видеопамять нулями
        mov     AL,0      ;черный цвет
@@NextPixels:
        mov     [GS:EDI],AL
        inc     EDI
        dec     ECX
        jnz     @@NextPixels
        popad
        ret
ENDP GClearScreen
ENDS
```

Программа Test256Mode, приведенная в листинге 4.4, демонстрирует вывод текста и графики (пучка прямых линий) в 256-цветном

режиме. В нем используются процедуры из листингов 4.2 и 4.3, а также универсальные модули из глав «Работа с клавиатурой» и «Недокументированные возможности процессоров Intel 80x86». При рисовании линий применяются упрощенные алгоритмы, каждый из которых позволяет чертить линию только под строго определенным углом.

В приведенном примере выбрано разрешение 640×480 точек, при котором текст, выводимый шрифтом 8×16, выглядит почти так же, как в текстовом режиме. Вы можете изменить разрешение, выбрав одно из трех предложенных в примере значений константы GraphicsMode. При увеличении разрешения становится менее заметным «лестничный эффект», то есть линии кажутся более «гладкими», но снижается яркость изображения (из-за того, что точки стали меньше, а линии — тоньше) и хуже читается текст (вследствие уменьшения размера символов). Изменяя номер режима, не забудьте присвоить соответствующие значения константам, описывающим физическое разрешение экрана (ScreenLength и ScreenHeigth).

Листинг 4.4. Вывод на экран текста и линий в режиме 256 цветов с разрешением 640×480

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Коды 256-цветных видеорежимов с линейной
; адресацией видеобуфера:
;    4101h - режим с разрешением 640x480
;    4103h - режим с разрешением 800x600
;    4105h - режим с разрешением 1024x768
GraphicsMode  equ  4101h
; Логическая ширина строки в пикселах
LogicalStringLength equ 1024
; Ширина экрана в пикселах
ScreenLength equ 640
; Высота экрана. строк
ScreenHeigth equ 480

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Абсолютный адрес начальной точки для "пучка"
```

```
; цветных линий
StartPixelAddress DD ?
; Текстовые сообщения
Txt1 DB 0,19
     DB "ТЕСТ 256-ЦВЕТНОГО РЕЖИМА ВИДЕОКОНТРОЛЛЕРА",0
Txt2 DB 2,22,"Вывод на экран текста, вертикальных,",0
     DB 3,1B
     DB "горизонтальных и диагональных цветных линий",0
AnyK DB 29,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC Test256Mode
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; Установить режим прямой адресации памяти
        call    Initialization
; "Захватить" текстовый шрифт
        call    GrabRusFont
; Установить видеорежим
        call    SetVESAVideoMode
; Отобразить текстовые сообщения
        ; Установить черный цвет фона
        mov     [DefaultBackground],BLACK
        ; Установить желтый цвет текста
        mov     [DefaultColor],YELLOW
        MGShowString Txt1
        ; Установить зеленый цвет текста
        mov     [DefaultColor],LIGHTGREEN
        MGShowText 2,Txt2
        ; Установить желтый цвет текста
        mov     [DefaultColor],YELLOW
        MGShowString AnyK

; Вычислить адрес начальной точки для пучка линий.
; Координаты точки (220,100)
        ; Умножить длину строки на номер строки (Y)
        mov     EAX,LogicalStringLength
        mov     EDX,100
```

продолжение ⇗

Листинг 4.4 (*продолжение*)

```
        mul     EDX
        ; Прибавить номер колонки (X)
        add     EAX,220
        ; Прибавить адрес видеобуфера
        add     EAX,[LinearVideoBuffer]
        ; Запомнить адрес начальной точки
        mov     [StartPixelAddress],EAX
; Начертить на экране горизонтальную белую
; линию с координатами (220,100)-(420,100)
        ; Установить цвет линии
        mov     AL,WHITE
        ; Записать адрес начальной
        ; точки в индексный регистр
        mov     EDI,[StartPixelAddress]
        ; Задать длину линии в пикселах
        mov     CX,200
@@L1:   mov     [GS:EDI],AL ;нарисовать пиксел
        inc     EDI         ;перейти в следующую позицию
        loop    @@L1
; Начертить вертикальную белую линию
; с координатами (220,100)-(220,300)
        mov     AL,WHITE
        mov     EDI,[StartPixelAddress]
        mov     CX,200
@@L2:   mov     [GS:EDI],AL
        add     EDI,LogicalStringLength
        loop    @@L2
; Начертить красную линию с координатами
; (220,100)-(420,300) (под углом 45 градусов
; к горизонтали)
        mov     AL,LIGHTRED
        mov     EDI,[StartPixelAddress]
        mov     CX,200
@@L3:   mov     [GS:EDI],AL
        add     EDI,LogicalStringLength+1
        loop    @@L3
; Начертить зеленую линию с координатами
; (220,100)-(420,200) (под углом 27 градусов
; к горизонтали)
        mov     AL,LIGHTGREEN
        mov     EDI,[StartPixelAddress]
        mov     CX,200/2
@@L4:   mov     [GS:EDI],AL
        inc     EDI
        mov     [GS:EDI],AL
        add     EDI,LogicalStringLength+1
        loop    @@L4
; Начертить синюю линию с координатами
```

```
; (220,100)-(320,300) (под углом 63 градуса
; к горизонтали)
        mov     AL,LIGHTBLUE
        mov     EDI,[StartPixelAddress]
        mov     CX,200/2
@@L5:   mov     [GS:EDI],AL
        add     EDI,LogicalStringLength
        mov     [GS:EDI],AL
        add     EDI,LogicalStringLength+1
        loop    @@L5
; Ожидать нажатия любой клавиши
        call    GetChar

; ВЫХОД ИЗ ПРОГРАММЫ
@@End:  ; Установить текстовый режим
        mov     AX,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP Test256Mode
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить набор процедур общего назначения,
; предназначенных для установки графических
; видеорежимов и работы в них
include "list4_02.inc"
; Подключить набор процедур вывода текста,
; предназначенных для 256-цветных режимов
include "list4_03.inc"

END
```

Примеры выполнения простых операций вывода для режимов группы HiColor16 (5:6:5) приведены в листингах 4.5 и 4.6: листинг 4.5 содержит процедуры вывода символа и очистки экрана для режима HiColor16, а листинг 4.6 — основную программу TestHiColorMode, выполняющую те же операции, что и пример из листинга 4.4.

ПРИМЕЧАНИЕ

Обратите внимание на особенность режимов HiColor16, усложняющую процесс программирования: для кодирования зеленого цвета используется шесть двоичных разрядов, а для красного и синего — по пять.

Листинг 4.5. Процедуры вывода символа и очистки экрана для режимов HiColor с линейной адресацией видеобуфера

```
DATASEG
; Цвет текста в графическом режиме по умолчанию
DefaultColor      DW  0FFFFh ;белый
; Цвет фона в графическом режиме по умолчанию
DefaultBackground DW  0       ;черный
ENDS

CODESEG
;*****************************************************
;* ВЫВОД СИМВОЛА 8x16 НА ЭКРАН В ГРАФИЧЕСКОМ РЕЖИМЕ *
;*           (для режимов HiColor 5:6:5)           *
;* Все параметры передаются через регистры:        *
;* AL - ASCII-код символа;                         *
;* DH - номер текстовой строки экрана;             *
;* DL - номер текстовой колонки экрана;            *
;* Используются цвет символов и цвет фона,         *
;* заданные по умолчанию.                          *
;*****************************************************
PROC PutGraChar NEAR
        push    DS
        pushad
        mov     CX,[CS:MainDataSeg]
        mov     DS,CX
        cld
        ; Смещение символа от начала шрифта
        mov     SI,offset Font8x16
        xor     AH,AH
        shl     AX,4
        add     SI,AX
        ; Вычислить левый верхний угол символа
        xor     EBX,EBX
        mov     BL,DH
        shl     EBX,15 ;умножить номер строки на 1024*32
        xor     DH,DH
        shl     DX,4   ;умножить номер столбца на 16
        or      BX,DX
        mov     EDI,EBX
        add     EDI,[LinearVideoBuffer]
        mov     BX,[DefaultColor]
        mov     DX,[DefaultBackground]

        mov     AH,16 ;счетчик строк маски буквы
@@M0:   lodsb
        mov     CX,8  ;счетчик точек в строке маски
@@M1:   rol     AL,1
        jc      @@M2
        mov     [GS:EDI],DX
```

```
        jmp     @@M3
@@M2:   mov     [GS:EDI],BX
@@M3:   add     EDI,2
        loop    @@M1
        add     EDI,2*LogicalStringLength-16
        dec     AH
        jnz     @@M0
; Завершение процедуры
@@EndPutGraChar:
        popad
        pop     DS
        inc     DL
        ret
ENDP PutGraChar

;**************************************
;* ОЧИСТКА ЭКРАНА В ГРАФИЧЕСКОМ РЕЖИМЕ *
;*   (процедура параметров не имеет)   *
;**************************************
PROC GClearScreen NEAR
        pushad
; Умножить высоту экрана ScreenHeigth на логическую
; ширину строки
        mov     ECX,ScreenHeigth
        shl     ECX,10
; Загрузить в индексный регистр линейный
; адрес видеопамяти
        mov     EDI,[LinearVideoBuffer]
; Заполнить видеопамять нулями
        mov     AX,0      ;черный цвет
@@NextPixels:
        mov     [GS:EDI],AX
        add     EDI,2
        dec     ECX
        jnz     @@NextPixels
        popad
        ret
ENDP GClearScreen
ENDS
```

Листинг 4.6. Вывод на экран текста и линий в режиме HiColor16 с разрешением 640×480

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Коды видеорежимов HiColor (5:6:5) с линейной
; адресацией видеобуфера:
;    4111h - режим с разрешением 640x480
```

продолжение ⇨

Листинг 4.6 (*продолжение*)

```
;    4114h - режим с разрешением 800x600
;    4117h - режим с разрешением 1024x768
GraphicsMode  equ  4111h
; Логическая ширина строки в пикселах
LogicalStringLength equ 1024
; Ширина экрана в пикселах
ScreenLength equ 640
; Высота экрана, строк
ScreenHeigth equ 480

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Абсолютный адрес начальной точки для "пучка"
; цветных линий
StartPixelAddress DD ?
; Текстовые сообщения
Txt1 DB 0,30,"ТЕСТ РЕЖИМА HICOLOR",0
Txt2 DB 2,22,"Вывод на экран текста, вертикальных,",0
     DB 3,18
     DB "горизонтальных и диагональных цветных линий",0
AnyK DB 29,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC TestHiColorMode
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; Установить режим прямой адресации памяти
        call    Initialization
; "Захватить" текстовый шрифт
        call    GrabRusFont
; Установить видеорежим
        call    SetVESAVideoMode
```

```
; Отобразить текстовые сообщения
        ; Установить черный цвет фона
        mov     [DefaultBackground],0
        ; Установить бирюзовый цвет текста
        mov     [DefaultColor],7FFh
        MGShowString Txt1
        ; Установить зеленый цвет текста
        mov     [DefaultColor],7E0h
        MGShowText 2,Txt2
        ; Установить желтый цвет текста
        mov     [DefaultColor],0FFE0h
        MGShowString AnyK

; Вычислить адрес начальной точки для пучка линий.
; Координаты точки (220,100)
        ; Умножить длину строки на номер строки (Y)
        mov     EAX,LogicalStringLength
        mov     EDX,100
        mul     EDX
        ; Прибавить номер колонки (X)
        add     EAX,220
        ; Умножить результат на 2
        shl     EAX,1
        ; Прибавить адрес видеобуфера
        add     EAX,[LinearVideoBuffer]
        ; Запомнить адрес начальной точки
        mov     [StartPixelAddress],EAX
; Начертить на экране горизонтальную белую
; линию с координатами (220,100)-(420,100)
        ; Установить цвет линии
        mov     AX,0FFFFh
        ; Записать адрес начальной
        ; точки в индексный регистр
        mov     EDI,[StartPixelAddress]
        ; Задать длину линии в пикселах
        mov     CX,200
@@L1:   mov     [GS:EDI],AX ;нарисовать пиксел
        add     EDI,2       ;перейти в следующую позицию
        loop    @@L1
; Начертить вертикальную белую линию
; с координатами (220,100)-(220,300)
        mov     AX,0FFFFh ;белый
        mov     EDI,[StartPixelAddress]
        mov     CX,200
@@L2:   mov     [GS:EDI],AX
        add     EDI,LogicalStringLength*2
        loop    @@L2
; Начертить красную линию с координатами
; (220,100)-(420,300) (под углом 45 градусов
; к горизонтали)
```

продолжение ⇗

Листинг 4.6 (*продолжение*)

```
        mov     AX,0F800h ;красный
        mov     EDI,[StartPixelAddress]
        mov     CX,200
@@L3:   mov     [GS:EDI],AX
        add     EDI,(LogicalStringLength+1)*2
        loop    @@L3
; Начертить зеленую линию с координатами
; (220,100)-(420,200) (под углом 27 градусов
; к горизонтали)
        mov     AX,7E0h   ;зеленый
        mov     EDI,[StartPixelAddress]
        mov     CX,200/2
@@L4:   mov     [GS:EDI],AX
        add     EDI,2
        mov     [GS:EDI],AX
        add     EDI,(LogicalStringLength+1)*2
        loop    @@L4
; Начертить синюю линию с координатами
; (220,100)-(320,300) (под углом 63 градуса
; к горизонтали)
        mov     AX,1Fh    ;синий
        mov     EDI,[StartPixelAddress]
        mov     CX,200/2
@@L5:   mov     [GS:EDI],AX
        add     EDI,LogicalStringLength*2
        mov     [GS:EDI],AX
        add     EDI,(LogicalStringLength+1)*2
        loop    @@L5
; Ожидать нажатия любой клавиши
        call    GetChar

; ВЫХОД ИЗ ПРОГРАММЫ
@@End:  ; Установить текстовый режим
        mov     AX,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP TestHiColorMode
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить набор процедур общего назначения,
; предназначенных для установки графических
```

```
; видеорежимов и работы в них
include "list4_02.inc"
; Подключить набор процедур вывода текста,
; предназначенных для режимов HiColor
include "list4_05.inc"

END
```

ПРИМЕЧАНИЕ

Для запуска примера из листинга 4.6 при разрешении 640×480 достаточно иметь видеоконтроллер с объемом памяти 1 Мбайт, а при более высоких разрешениях нужно 2 Мбайт.

Пример для режимов группы TrueColor32 (8:8:8:8) также состоит из двух модулей: листинг 4.7 содержит процедуры вывода символа и очистки экрана, а листинг 4.8 — основную программу TestTrueColorMode. Поскольку заранее *неизвестно*, поддерживает ли видеоконтроллер режим TrueColor32, и какой номер присвоен изготовителями контроллера этому режиму, приходится проводить поиск по всему списку режимов. Поиск осуществляется по заданным значениям констант ScreenLength и ScreenHeigth.

Листинг 4.7. Процедуры вывода символа и очистки экрана для режимов TrueColor32 с линейной адресацией видеобуфера

```
DATASEG
; Цвет текста в графическом режиме по умолчанию
DefaultColor      DD  0FFFFFFh ;белый
; Цвет фона в графическом режиме по умолчанию
DefaultBackground DD  0        ;черный
ENDS

CODESEG
;****************************************************
;* ВЫВОД СИМВОЛА 8x16 НА ЭКРАН В ГРАФИЧЕСКОМ РЕЖИМЕ *
;*            (для режимов HiColor 5:6:5)           *
;* Все параметры передаются через регистры:         *
;* AL - ASCII-код символа;                          *
;* DH - номер текстовой строки экрана;              *
;* DL - номер текстовой колонки экрана;             *
;* Используются цвет символов и цвет фона,          *
;* заданные по умолчанию.                           *
;****************************************************
PROC PutGraChar NEAR
        push  DS
        pushad
        mov   CX,[CS:MainDataSeg]
        mov   DS,CX
```

продолжение ⇒

Листинг 4.7 (*продолжение*)

```
        cld
        ; Смещение символа от начала шрифта
        mov     SI,offset Font8x16
        xor     AH,AH
        shl     AX,4
        add     SI,AX
        ; Вычислить левый верхний угол символа
        xor     EBX,EBX
        mov     BL,DH
        shl     EBX,16 ;умножить номер строки на 1024*64
        xor     DH,DH
        shl     DX,5   ;умножить номер столбца на 32
        or      BX,DX
        mov     EDI,EBX
        add     EDI,[LinearVideoBuffer]
        mov     EBX,[DefaultColor]
        mov     EDX,[DefaultBackground]

        mov     AH,16 ;счетчик строк маски буквы
@@M0:   lodsb
        mov     CX,8  ;счетчик точек в строке маски
@@M1:   rol     AL,1
        jc      @@M2
        mov     [GS:EDI],EDX
        jmp     @@M3
@@M2:   mov     [GS:EDI],EBX
@@M3:   add     EDI,4
        loop    @@M1
        add     EDI,4*LogicalStringLength-32
        dec     AH
        jnz     @@M0
; Завершение процедуры
@@EndPutGraChar:
        popad
        pop     DS
        inc     DL
        ret
ENDP PutGraChar

;****************************************
;* ОЧИСТКА ЭКРАНА В ГРАФИЧЕСКОМ РЕЖИМЕ *
;*   (процедура параметров не имеет)   *
;****************************************
PROC GClearScreen NEAR
        pushad
; Умножить высоту экрана ScreenHeigth на логическую
; ширину строки
        mov     ECX,ScreenHeigth
        shl     ECX,10
```

```
; Загрузить в индексный регистр линейный
; адрес видеопамяти
        mov     EDI,[LinearVideoBuffer]
; Заполнить видеопамять нулями
        mov     EAX,0      ;черный цвет
@@NextPixels:
        mov     [GS:EDI],EAX
        add     EDI,4
        dec     ECX
        jnz     @@NextPixels
        popad
        ret
ENDP GClearScreen
ENDS
```

Листинг 4.8. Вывод на экран текста и линий в режиме TrueColor32 с разрешением 640×480

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Номер видеорежима заранее не известен
GraphicsMode equ 0
; Логическая ширина строки в пикселах
LogicalStringLength equ 1024
; Ширина экрана в пикселах
ScreenLength equ 640
; Высота экрана, строк
ScreenHeigth equ 480

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Абсолютный адрес начальной точки для "пучка"
; цветных линий
StartPixelAddress DD ?
; Текстовые сообщения
Txt1 DB 0,28,"ТЕСТ РЕЖИМА TRUECOLOR32",0
Txt2 DB 2,22,"Вывод на экран текста, вертикальных,",0
     DB 3,18
     DB "горизонтальных и диагональных цветных линий",0
AnyK DB 29,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
```

продолжение

Листинг 4.8 (*продолжение*)

```
DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC TestTrueColorMode
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; "Захватить" текстовый шрифт
        call    GrabRusFont
; Установить видеорежим
        call    SetTrueColor32
; Установить режим прямой адресации памяти
        call    Initialization
; Отобразить текстовые сообщения
        ; Установить черный цвет фона
        mov     [dword ptr DefaultBackground],0
        ; Установить фиолетовый цвет текста
        mov     [dword ptr DefaultColor],0FF00FFh
        MGShowString Txt1
        ; Установить зеленый цвет текста
        mov     [dword ptr DefaultColor],0FF00h
        MGShowText 2,Txt2
        ; Установить желтый цвет текста
        mov     [dword ptr DefaultColor],0FFFF00h
        MGShowString AnyK

; Вычислить адрес начальной точки для пучка линий.
; Координаты точки (220,100)
        ; Умножить длину строки на номер строки (Y)
        mov     EAX,LogicalStringLength
        mov     EDX,100
        mul     EDX
        ; Прибавить номер колонки (X)
        add     EAX,220
        ; Умножить результат на 4
        shl     EAX,2
        ; Прибавить адрес видеобуфера
        add     EAX,[LinearVideoBuffer]
        ; Запомнить адрес начальной точки
        mov     [StartPixelAddress],EAX
; Начертить на экране горизонтальную белую
; линию с координатами (220,100)-(420,100)
```

```
        ; Установить цвет линии
        mov     EAX,0FFFFFFh ;белый
        ; Записать адрес начальной
        ; точки в индексный регистр
        mov     EDI,[StartPixelAddress]
        ; Задать длину линии в пикселах
        mov     CX,200
@@L1:   mov     [GS:EDI],EAX ;нарисовать пиксел
        add     EDI,4        ;перейти в следующую позицию
        loop    @@L1
; Начертить вертикальную белую линию
; с координатами (220,100)-(220,300)
        mov     EAX,0FFFFFFh ;белый
        mov     EDI,[StartPixelAddress]
        mov     CX,200
@@L2:   mov     [GS:EDI],EAX
        add     EDI,LogicalStringLength*4
        loop    @@L2
; Начертить красную линию с координатами
; (220,100)-(420,300) (под углом 45 градусов
; к горизонтали)
        mov     EAX,0FF0000h ;красный
        mov     EDI,[StartPixelAddress]
        mov     CX,200
@@L3:   mov     [GS:EDI],EAX
        add     EDI,(LogicalStringLength+1)*4
        loop    @@L3
; Начертить зеленую линию с координатами
; (220,100)-(420,200) (под углом 27 градусов
; к горизонтали)
        mov     EAX,0FF00h   ;зеленый
        mov     EDI,[StartPixelAddress]
        mov     CX,200/2
@@L4:   mov     [GS:EDI],EAX
        add     EDI,4
        mov     [GS:EDI],EAX
        add     EDI,(LogicalStringLength+1)*4
        loop    @@L4
; Начертить синюю линию с координатами
; (220,100)-(320,300) (под углом 63 градуса
; к горизонтали)
        mov     EAX,0FFh     ;синий
        mov     EDI,[StartPixelAddress]
        mov     CX,200/2
@@L5:   mov     [GS:EDI],EAX
        add     EDI,LogicalStringLength*4
        mov     [GS:EDI],EAX
        add     EDI,(LogicalStringLength+1)*4
        loop    @@L5
; Ожидать нажатия любой клавиши
        call    GetChar
```

продолжение ↗

Листинг 4.8 (*продолжение*)

```
; ВЫХОД ИЗ ПРОГРАММЫ
@@End:  ; Установить текстовый режим
        mov    AX,3
        int    10h
        ; Выход в DOS
        mov    AH,4Ch
        int    21h
ENDP TestTrueColorMode
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить набор процедур общего назначения,
; предназначенных для установки графических
; видеорежимов и работы в них
include "list4_02.inc"
; Подключить набор процедур вывода текста,
; предназначенных для режимов TrueColor32
include "list4_07.inc"

END
```

ПРИМЕЧАНИЕ

Для запуска примера из листинга 4.8 необходим видеоконтроллер с объемом памяти не менее 4 Мбайт (в старых контроллерах с объемом памяти 1–2 Мбайт в целях зкономии вместо TrueColor32 использовался режим TrueColor24).

Если сравнить примеры для разных типов видеорежимов, то становится очевидным, что при выводе текста и чертежей режимы HiColor и TrueColor не дают никаких преимуществ по сравнению с 256-цветными режимами. В то же время 256-цветные режимы экономно используют видеопамять (достаточно 1–2 Мбайт даже при высоком разрешении) и существенно превосходят все остальные типы режимов по скорости вывода информации (правда, только при использовании специальных алгоритмов, позволяющих выводить по четыре пиксела за одну операцию).

Пример использования алгоритма Брезенхема для рисования линии в 256-цветных режимах приведен в листингах 4.9 и 4.10. Процедура рисования линии EVGALine, приведенная в листинге 4.9, получена путем прямого перевода примера из книги Майкла Абраша [1]

с языка С на язык Ассемблер x86. Процедура использует две вспомогательные подпрограммы Octant0 и Octant1 для рисования линий в различных октантах. Процедура специально приведена в *неоптимизированном* варианте (переменные хранятся в памяти, а не в регистрах), так как после оптимизации программа утрачивает свойство *наглядности*. Вызывающая (основная) программа TestLines256 из листинга 4.10 предназначена для тестирования процедуры рисования линии: она выводит четыре пучка линий различных цветов — белого, красного, синего и зеленого.

Листинг 4.9. Подпрограмма рисования линии по алгоритму Брезенхема для 256-цветных режимов

```
DATASEG
EVEN        ;выравнивание смещения данных на 2
; Координаты начала линии
X0          DD ?
Y0          DD ?
; Координаты конца линии
X1          DD ?
Y1          DD ?
; Длина линии по X и по Y
DeltaX      DD ?
DeltaY      DD ?
; Направление рисования по X (1 - линия
; прорисовывается слева направо, -1 - справа налево)
XDirection DD ?
; Цвет линии
Color       DB ?
EVEN        ;выравнивание смещения данных на 2
; Внутренние переменные процедур рисования
DeltaYx2                DD ?
DeltaYx2MinusDeltaXx2   DD ?
DeltaXx2                DD ?
DeltaXx2MinusDeltaYx2   DD ?
ErrorTerm               DD ? ;ошибка накопления
PixelOffset             DD ?
ENDS

CODESEG
;*********************************************
;* ПРОЦЕДУРА РИСОВАНИЯ ЛИНИИ В ОКТАНТАХ 0 И 3 *
;*          (|DeltaX| >= DeltaY)             *
;*********************************************
PROC Octant0 NEAR
        pushad
; Установить начальную ошибку накопления и значения,
; используемые во внутреннем цикле
```

Листинг 4.9 (*продолжение*)

```
        ; (DeltaYx2 = 2*DeltaY)
        mov     EAX,[DeltaY]
        shl     EAX,1
        mov     [DeltaYx2],EAX
        ; (ErrorTerm = DeltaYx2 - DeltaX)
        sub     EAX,[DeltaX]
        mov     [ErrorTerm],EAX
        ; (DeltaYx2MinusDeltaXx2 = DeltaYx2 - 2*DeltaX)
        sub     EAX,[DeltaX]
        mov     [DeltaYx2MinusDeltaXx2],EAX
; Рисуем линию
        ; (PixelOffset=Y0*LogicalStringLength+X0)
        mov     EAX,[Y0]
        mov     EDX,LogicalStringLength
        mul     EDX
        add     EAX,[X0]
        mov     [PixelOffset],EAX
; Рисуем первую точку линии
        mov     EBX,[PixelOffset]
        add     EBX,[LinearVideoBuffer]
        mov     AL,[Color]
        mov     [GS:EBX],AL
; Цикл, пока DeltaX>=0
@@NextDot:
        ; Проверить, не пора ли перейти на точку
        ; по оси Y
        cmp     [ErrorTerm],0
        jl      @@AddError
        ; Сделать шаг по Y
        add     [PixelOffset],LogicalStringLength
        ; Увеличить ошибку накопления
        mov     EAX,[DeltaYx2MinusDeltaXx2]
        add     [ErrorTerm],EAX
        jmp     @@PutPixel
@@AddError:
        ; Увеличить ошибку накопления
        mov     EAX,[DeltaYx2]
        add     [ErrorTerm],EAX
@@PutPixel:
        ; Сделать шаг по X
        mov     EAX,[XDirection]
        add     [PixelOffset],EAX
        ; Вывести очередную точку линии на экран
        mov     EBX,[PixelOffset]
        add     EBX,[LinearVideoBuffer]
        mov     AL,[Color]
        mov     [GS:EBX],AL
        dec     [DeltaX]
        jnz     @@NextDot
```

```
        popad
        ret
ENDP Octant0

;*********************************************
;* ПРОЦЕДУРА РИСОВАНИЯ ЛИНИИ В ОКТАНТАХ 1 И 2 *
;*           (|DeltaX| < DeltaY)              *
;*********************************************
PROC Octant1 NEAR
        pushad
; Установить начальную ошибку накопления и значения,
; используемые во внутреннем цикле
        ; (DeltaXx2 = 2*DeltaX)
        mov     EAX,[DeltaX]
        shl     EAX,1
        mov     [DeltaXx2],EAX
        ; (ErrorTerm = DeltaXx2 - DeltaY)
        sub     EAX,[DeltaY]
        mov     [ErrorTerm],EAX
        ; (DeltaXx2MinusDeltaYx2 = DeltaXx2 - 2*DeltaY)
        sub     EAX,[DeltaY]
        mov     [DeltaXx2MinusDeltaYx2],EAX
; Рисуем линию
        ; (PixelOffset=Y0*LogicalStringLength+X0);
        mov     EAX,[Y0]
        mov     EDX,LogicalStringLength
        mul     EDX
        add     EAX,[X0]
        mov     [PixelOffset],EAX
; Рисуем первый пиксел
        mov     EBX,[PixelOffset]
        add     EBX,[LinearVideoBuffer]
        mov     AL,[Color]
        mov     [GS:EBX],AL
; Цикл, пока DeltaY>=0
@@NextDot:
        ; Проверить, не пора ли перейти на точку
        ; по оси X
        cmp     [ErrorTerm],0
        jl      @@AddError
        ; Сделать шаг по X
        mov     EAX,[XDirection]
        add     [PixelOffset],EAX
        ; Увеличить ошибку накопления
        mov     EAX,[DeltaXx2MinusDeltaYx2]
        add     [ErrorTerm],EAX
        jmp     @@PutPixel
        ; Увеличить ошибку накопления
@@AddError:
        mov     EAX,[DeltaXx2]
```

продолжение ⇨

Листинг 4.9 (*продолжение*)

```
        add     [ErrorTerm],EAX
@@PutPixel:
        ; Сделать шаг по Y
        add     [PixelOffset],LogicalStringLength
        ; Вывести очередную точку линии на экран
        mov     EBX,[PixelOffset]
        add     EBX,[LinearVideoBuffer]
        mov     AL,[Color]
        mov     [GS:EBX],AL
        dec     [DeltaY]
        jnz     @@NextDot
        popad
        ret
ENDP Octant1

;*****************************************************
;* ПРОЦЕДУРА РИСОВАНИЯ ЛИНИИ ПО АЛГОРИТМУ БРЕЗЕНХЕМА *
;* Передача параметров выполняется через           *
;* глобальные переменные:                          *
;* X0, Y0 - координаты начальной точки;            *
;* X1, Y1 - координаты конечной точки;             *
;* Color - цвет линии.                             *
;*****************************************************
PROC EVGALine NEAR
        pushad
; Запомнить координаты линии в стеке
        push    [X0]
        push    [Y0]
        push    [X1]
        push    [Y1]
; Если Y0 > Y1, поменять местами начальную
; и конечную точки линии.
        mov     EAX,[Y0]
        cmp     EAX,[Y1]
        jbe     @@L0
        xchg    EAX,[Y1]
        xchg    EAX,[Y0]
        mov     EAX,[X0]
        xchg    EAX,[X1]
        xchg    EAX,[X0]
@@L0:
; Вычислить DeltaX
        mov     EAX,[X1]
        sub     EAX,[X0]
        mov     [DeltaX],EAX
; Вычислить DeltaY
        mov     EAX,[Y1]
        sub     EAX,[Y0]
        mov     [DeltaY],EAX
```

```
; Выбрать номер октанта и направление движения
        mov     [XDirection],1
        mov     EAX,[DeltaX]
        cmp     EAX,0
        jge     @@L1
        neg     EAX
        mov     [DeltaX],EAX
        mov     [XDirection],-1
@@L1:   cmp     EAX,[DeltaY]
        jle     @@L2
        call    Octant0
        jmp     @@End
@@L2:   call    Octant1

; Восстановить координаты линии
@@End:  pop    [Y1]
        pop    [X1]
        pop    [Y0]
        pop    [X0]
        popad
        ret
ENDP EVGALine
ENDS
```

Листинг 4.10. Рисование линий по алгоритму Брезенхема в режиме 256 цветов с разрешением 640×480

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Коды 256-цветных видеорежимов с линейной
; адресацией видеобуфера:
;    4101h - режим с разрешением 640x480
;    4103h - режим с разрешением 800x600
;    4105h - режим с разрешением 1024x768
GraphicsMode  equ  4101h
; Логическая ширина строки в пикселах
LogicalStringLength equ 1024
; Ширина экрана в пикселах
ScreenLength equ 640
; Высота экрана, строк
ScreenHeigth equ 480

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"
```

продолжение ⇗

Листинг 4.10 (*продолжение*)

```
DATASEG
; Текстовые сообщения
Txt1 DB 0,22,"РИСОВАНИЕ ЛИНИЙ В 256-ЦВЕТНОМ РЕЖИМЕ",0
AnyK DB 29,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC TestLines256
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; Установить режим прямой адресации памяти
        call    Initialization
; "Захватить" текстовый шрифт
        call    GrabRusFont
; Установить видеорежим
        call    SetVESAVideoMode
; Отобразить текстовые сообщения
        ; Установить черный цвет фона
        mov     [DefaultBackground],BLACK
        ; Установить зеленый цвет текста
        mov     [DefaultColor],LIGHTCYAN
        MGShowString Txt1
        ; Установить желтый цвет текста
        mov     [DefaultColor],YELLOW
        MGShowString AnyK

; Занести координаты начальной точки для пучка линий
        mov     [dword ptr X0],320
        mov     [dword ptr Y0],240
; Нарисовать пучок белых линий
        mov     EAX,[dword ptr X0]
        add     EAX,100
        mov     [dword ptr X1],EAX
        mov     EAX,[dword ptr Y0]
        sub     EAX,100
        mov     [dword ptr Y1],EAX
        mov     [Color],WHITE
        mov     CX,50
@@NextWhiteLine:
```

```
        call    EVGALine
        add     [dword ptr Y1],4
        loop    @@NextWhiteLine
; Нарисовать пучок красных линий
        mov     EAX,[dword ptr X0]
        add     EAX,100
        mov     [dword ptr X1],EAX
        mov     EAX,[dword ptr Y0]
        add     EAX,100
        mov     [dword ptr Y1],EAX
        mov     [Color],LIGHTRED
        mov     CX,50
@@NextRedLine:
        call    EVGALine
        sub     [dword ptr X1],4
        loop    @@NextRedLine
; Нарисовать пучок зеленых линий
        mov     EAX,[dword ptr X0]
        sub     EAX,100
        mov     [dword ptr X1],EAX
        mov     EAX,[dword ptr Y0]
        add     EAX,100
        mov     [dword ptr Y1],EAX
        mov     [Color],LIGHTGREEN
        mov     CX,50
@@NextGreenLine:
        call    EVGALine
        sub     [dword ptr Y1],4
        loop    @@NextGreenLine
; Нарисовать пучок синих линий
        mov     EAX,[dword ptr X0]
        sub     EAX,100
        mov     [dword ptr X1],EAX
        mov     EAX,[dword ptr Y0]
        sub     EAX,100
        mov     [dword ptr Y1],EAX
        mov     [Color],LIGHTBLUE
        mov     CX,50
@@NextBlueLine:
        call    EVGALine
        add     [dword ptr X1],4
        loop    @@NextBlueLine

; Ожидать нажатия любой клавиши
        call    GetChar
; Установить текстовый режим
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
```

продолжение ⇨

Листинг 4.10 (*продолжение*)

```
        int     21h
ENDP TestLines256
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить набор процедур общего назначения,
; предназначенных для установки графических
; видеорежимов и работы в них
include "list4_02.inc"
; Подключить набор процедур вывода текста,
; предназначенных для 256-цветных режимов
include "list4_03.inc"
; Подключить подпрограмму рисования линии по
; алгоритму Брезенхема для 256-цветных режимов
include "list4_09.inc"

END
```

Пример использования алгоритма Брезенхема в режимах TrueColor32 дан в листингах 4.11 и 4.12. Приведенная в листинге 4.11 процедура рисования линии EVGALine по сути та же самая, что и в листинге 4.9, но адаптирована для режимов TrueColor. Программа TestLinesTrueColor32, приведенная в листинге 4.12, не только тестирует процедуру рисования линии, но и демонстрирует один из приемов создания спецэффектов: при выводе пучка линий плавно меняется яркость, а линии размещаются впритирку друг к другу, создавая в результате иллюзию пропеллера.

ПРИМЕЧАНИЕ

Как уже было указано выше, для запуска программ, работающих в режиме TrueColor32, необходим видеоконтроллер, имеющий не менее 4 Мбайт памяти.

Листинг 4.11. Подпрограмма рисования линии по алгоритму Брезенхема для режима TrueColor32

```
DATASEG
EVEN        ;выравнивание смещения данных на 2
; Координаты начала линии
X0          DD ?
Y0          DD ?
; Координаты конца линии
```

```
X1          DD ?
Y1          DD ?
; Длина линии по X и по Y
DeltaX      DD ?
DeltaY      DD ?
; Направление рисования по X (1 - линия
; прорисовывается слева направо, -1 - справа налево)
XDirection DD ?
; Цвет линии
Color       DD ?
EVEN        ;выравнивание смещения данных на 2
; Внутренние переменные процедур рисования
DeltaYx2                DD ?
DeltaYx2MinusDeltaXx2   DD ?
DeltaXx2                DD ?
DeltaXx2MinusDeltaYx2   DD ?
ErrorTerm               DD ? ;ошибка накопления
PixelOffset             DD ?
ENDS

CODESEG
;*********************************************
;* ПРОЦЕДУРА РИСОВАНИЯ ЛИНИИ В ОКТАНТАХ 0 И 3 *
;*            (|DeltaX| >= DeltaY)           *
;*********************************************
PROC Octant0 NEAR
        pushad
; Установить начальную ошибку накопления и значения,
; используемые во внутреннем цикле
        ; (DeltaYx2 = 2*DeltaY)
        mov     EAX,[DeltaY]
        shl     EAX,1
        mov     [DeltaYx2],EAX
        ; (ErrorTerm = DeltaYx2 - DeltaX)
        sub     EAX,[DeltaX]
        mov     [ErrorTerm],EAX
        ; (DeltaYx2MinusDeltaXx2 = DeltaYx2 - 2*DeltaX)
        sub     EAX,[DeltaX]
        mov     [DeltaYx2MinusDeltaXx2],EAX
; Рисуем линию
        ; (PixelDffset=4*(Y0*LogicalStringLength+X0))
        mov     EAX,[Y0]
        mov     EDX,LogicalStringLength
        mul     EDX
        add     EAX,[X0]
        shl     EAX,2
        mov     [PixelOffset],EAX
; Рисуем первую точку линии
        mov     EBX,[PixelOffset]
        add     EBX,[LinearVideoBuffer]
        mov     EAX,[Color]
```

продолжение ⇨

Листинг 4.11 (*продолжение*)

```
        mov     [GS:EBX],EAX
; Цикл, пока DeltaX>=0
@@NextDot:
        ; Проверить, не пора ли перейти на точку
        ; по оси Y
        cmp     [ErrorTerm],0
        jl      @@AddError
        ; Сделать шаг по Y
        add     [PixelOffset],LogicalStringLength*4
        ; Увеличить ошибку накопления
        mov     EAX,[DeltaYx2MinusDeltaXx2]
        add     [ErrorTerm],EAX
        jmp     @@PutPixel
@@AddError:
        ; Увеличить ошибку накопления
        mov     EAX,[DeltaYx2]
        add     [ErrorTerm],EAX
@@PutPixel:
        ; Сделать шаг по X
        mov     EAX,[XDirection]
        add     [PixelOffset],EAX
        ; Вывести очередную точку линии на экран
        mov     EBX,[PixelOffset]
        add     EBX,[LinearVideoBuffer]
        mov     EAX,[Color]
        mov     [GS:EBX],EAX
        dec     [DeltaX]
        jnz     @@NextDot
        popad
        ret
ENDP Octant0

;*********************************************
;* ПРОЦЕДУРА РИСОВАНИЯ ЛИНИИ В ОКТАНТАХ 1 И 2 *
;*           (|DeltaX| < DeltaY)             *
;*********************************************
PROC Octant1 NEAR
        pushad
; Установить начальную ошибку накопления и значения,
; используеиые во внутреннеи цикле
        ; (DeltaXx2 = 2*DeltaX)
        mov     EAX,[DeltaX]
        shl     EAX,1
        mov     [DeltaXx2],EAX
        ; (ErrorTerm = DeltaXx2 - DeltaY)
        sub     EAX,[DeltaY]
        mov     [ErrorTerm],EAX
        ; (DeltaXx2MinusDeltaYx2 = DeltaXx2 - 2*DeltaY)
        sub     EAX,[DeltaY]
```

```
        mov     [DeltaXx2MinusDeltaYx2],EAX
; Рисуем линию
        ; (PixelOffset=4*(Y0*LogicalStringLength+X0));
        mov     EAX,[Y0]
        mov     EDX,LogicalStringLength
        mul     EDX
        add     EAX,[X0]
        shl     EAX,2
        mov     [PixelOffset],EAX
; Рисуем первый пиксел
        mov     EBX,[PixelOffset]
        add     EBX,[LinearVideoBuffer]
        mov     EAX,[Color]
        mov     [GS:EBX],EAX
; Цикл, пока DeltaY>=0
@@NextDot:
        ; Проверить, не пора ли перейти на точку
        ; по оси X
        cmp     [ErrorTerm],0
        jl      @@AddError
        ; Сделать шаг по X
        mov     EAX,[XDirection]
        add     [PixelOffset],EAX
        ; Увеличить ошибку накопления
        mov     EAX,[DeltaXx2MinusDeltaYx2]
        add     [ErrorTerm],EAX
        jmp     @@PutPixel
        ; Увеличить ошибку накопления
@@AddError:
        mov     EAX,[DeltaXx2]
        add     [ErrorTerm],EAX
@@PutPixel:
        ; Сделать шаг по Y
        add     [PixelOffset],LogicalStringLength*4
        ; Вывести очередную точку линии на экран
        mov     EBX,[PixelOffset]
        add     EBX,[LinearVideoBuffer]
        mov     EAX,[Color]
        mov     [GS:EBX],EAX
        dec     [DeltaY]
        jnz     @@NextDot
        popad
        ret
ENDP Octant1

;*****************************************************
;* ПРОЦЕДУРА РИСОВАНИЯ ЛИНИИ ПО АЛГОРИТМУ БРЕЗЕНХЕМА *
;* Передача параметров выполняется через             *
;* глобальные переменные:                            *
;* X0, Y0 - координаты начальной точки;              *
```

продолжение ⇨

Листинг 4.11 (*продолжение*)

```
;* X1, Y1 - координаты конечной точки;            *
;* Color - цвет линии.                             *
;**************************************************
PROC EVGALine NEAR
        pushad
; Запомнить координаты линии в стеке
        push    [X0]
        push    [Y0]
        push    [X1]
        push    [Y1]
; Если Y0 > Y1, поменять местами начальную
; и конечную точки линии.
        mov     EAX,[Y0]
        cmp     EAX,[Y1]
        jbe     @@L0
        xchg    EAX,[Y1]
        xchg    EAX,[Y0]
        mov     EAX,[X0]
        xchg    EAX,[X1]
        xchg    EAX,[X0]
@@L0:
; Вычислить DeltaX
        mov     EAX,[X1]
        sub     EAX,[X0]
        mov     [DeltaX],EAX
; Вычислить DeltaY
        mov     EAX,[Y1]
        sub     EAX,[Y0]
        mov     [DeltaY],EAX

; Выбрать номер октанта и направление движения
        mov     [XDirection],1*4
        mov     EAX,[DeltaX]
        cmp     EAX,0
        jge     @@L1
        neg     EAX
        mov     [DeltaX],EAX
        mov     [XDirection],-1*4
@@L1:   cmp     EAX,[DeltaY]
        jle     @@L2
        call    Octant0
        jmp     @@End
@@L2:   call    Octant1

; Восстановить координаты линии
@@End:  pop     [Y1]
        pop     [X1]
        pop     [Y0]
        pop     [X0]
```

```
        popad
        ret
ENDP EVGALine
ENDS
```

Листинг 4.12. Рисование линий по алгоритму Брезенхема в режиме TrueColor32 с разрешением 640×480

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Номер видеорежима заранее не известен
GraphicsMode equ 0
; Логическая ширина строки в пикселах
LogicalStringLength equ 1024
; Ширина экрана в пикселах
ScreenLength equ 640
; Высота экрана, строк
ScreenHeigth equ 480

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Текстовые сообщения
Txt1 DB 0,23,"РИСОВАНИЕ ЛИНИЙ В РЕЖИМЕ TRUECOLOR",0
AnyK DB 29,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC TestLinesTrueColor32
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; "Захватить" текстовый шрифт
        call    GrabRusFont
```

продолжение ⇨

Листинг 4.12 (*продолжение*)

```
; Установить видеорежим
        call    SetTrueColor32
; Установить режим прямой адресации памяти
        call    Initialization
; Отобразить текстовые сообщения
        ; Установить черный цвет фона
        mov     [dword ptr DefaultBackground],0
        ; Установить темно-желтый цвет текста
        mov     [dword ptr DefaultColor],0E0B000h
        MGShowString Txt1
        ; Установить желтый цвет текста
        mov     [dword ptr DefaultColor],0FFFF00h
        MGShowString AnyK

; Занести координаты начальной точки для пучка линий
        mov     [dword ptr X0],320
        mov     [dword ptr Y0],240
; Нарисовать пучок белых линий
        mov     EAX,[dword ptr X0]
        add     EAX,127
        mov     [dword ptr X1],EAX
        mov     EAX,[dword ptr Y0]
        sub     EAX,127
        mov     [dword ptr Y1],EAX
        mov     [dword ptr Color],0 ;0FFFFFFh
        mov     CX,254
@@NextWhiteLine:
        call    EVGALine
        add     [dword ptr Y1],1
        add     [dword ptr Color],010101h
        loop    @@NextWhiteLine
; Нарисовать пучок красных линий
        mov     EAX,[dword ptr X0]
        add     EAX,127
        mov     [dword ptr X1],EAX
        mov     EAX,[dword ptr Y0]
        add     EAX,127
        mov     [dword ptr Y1],EAX
        mov     [dword ptr Color],0 ;0FF0000h
        mov     CX,254
@@NextRedLine:
        call    EVGALine
        sub     [dword ptr X1],1
        add     [dword ptr Color],010000h
        loop    @@NextRedLine
; Нарисовать пучок зеленых линий
        mov     EAX,[dword ptr X0]
        sub     EAX,127
```

```
        mov     [dword ptr X1],EAX
        mov     EAX,[dword ptr Y0]
        add     EAX,127
        mov     [dword ptr Y1],EAX
        mov     [dword ptr Color],0 ;0FF00h
        mov     CX,254
@@NextGreenLine:
        call    EVGALine
        sub     [dword ptr Y1],1
        add     [dword ptr Color],0100h
        loop    @@NextGreenLine
; Нарисовать пучок синих линий
        mov     EAX,[dword ptr X0]
        sub     EAX,127
        mov     [dword ptr X1],EAX
        mov     EAX,[dword ptr Y0]
        sub     EAX,127
        mov     [dword ptr Y1],EAX
        mov     [dword ptr Color],0; 0FFh
        mov     CX,254
@@NextBlueLine:
        call    EVGALine
        add     [dword ptr X1],1
        add     [dword ptr Color],1
        loop    @@NextBlueLine

; Ожидать нажатия любой клавиши
        call    GetChar
; Установить текстовый режим
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP TestLinesTrueColor32
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить набор процедур общего назначения,
; предназначенных для установки графических
; видеорежимов и работы в них
include "list4_02.inc"
; Подключить набор процедур вывода текста,
; предназначенных для режимов TrueColor32
include "list4_07.inc"
```

продолжение ⇨

Листинг 4.12 (*продолжение*)

```
; Подключить подпрограмму рисования линии по
; алгоритму Брезенхема для режимов TrueColor32
include "list4_11.inc"

END
```

Масштабирование изображений

При работе с различными изображениями на компьютере довольно часто возникает необходимость в их увеличении или уменьшении, то есть в масштабировании. Алгоритмы для получения изображения в произвольном масштабе относительно сложные, но увеличить или уменьшить изображение в целое число раз нетрудно.

Чтобы увеличить изображение без искажений, достаточно просто увеличить в N раз каждую его точку, то есть представить ее в виде квадрата N×N точек. Недостаток такого способа заключается в том, что сильно проявляет себя «лестничный эффект» — становится очень заметной ступенчатость изображения. Пример двукратного увеличения изображения текстового символа показан на рис. 4.13.

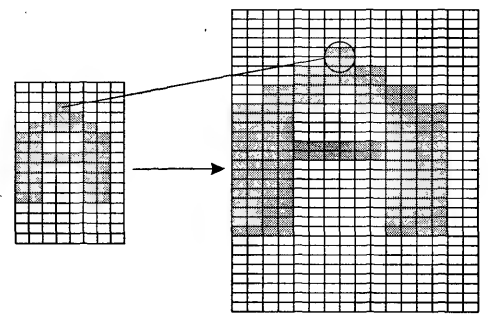


Рис. 4.13. Увеличение размера изображения символа в два раза

Столь же просто можно деформировать изображение, растягивая его по вертикали и горизонтали в различное число раз. При этом точка превращается уже не в квадрат, а в прямоугольник, содержащий N×M точек.

Чтобы сжать изображение в целое число раз, нужно вычислить среднюю яркость квадрата из N×N точек и округлить ее до ближайшего

доступного значения яркости. Однако такое округление приводит к сильным искажениям при выводе черно-белого изображения (рис. 4.14, *слева*).

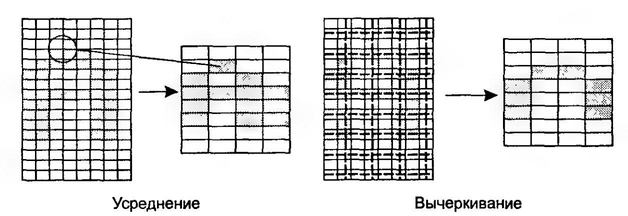


Рис. 4.14. Сжатие изображения в два раза различными способами

Другой простой способ сжатия — вычеркивание. В этом случае сохраняются лишь одна из N последовательно расположенных строк и одна из N колонок, а остальные зачеркиваются (рис. 4.14, *справа*). Основной недостаток данного способа состоит в том, что тонкие вертикальные и горизонтальные линии могут быть полностью потеряны: в приведенном примере таким образом стерта перекладина у буквы А.

Программа ShowFont, предназначенная для просмотра в увеличенном масштабе символов шрифта, извлеченного из знакогенератора видеоконтроллера, приведена в листинге 4.13. Программа использует две вспомогательные процедуры:

- процедура ShowRusFont отображает на экран шрифт в виде таблицы в масштабе 1:1 — каждому биту маски соответствует на экране один пиксел;
- процедура ShowLargeChar осуществляет вывод символа в верхней части экрана (по центру) в увеличенном масштабе — каждому биту маски соответствует на экране квадрат размером 8×8 пикселов.

Листинг 4.13. Просмотр символов шрифта 8×16 в графическом режиме в увеличенном масштабе

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Код видеорежиыа 640x480, 256 цветов, с линейной
; адресацией видеобуфера
```

продолжение ⇒

Листинг 4.13 (*продолжение*)

```
GraphicsMode equ 4101h
; Логическая ширина строки в пикселах
LogicalStringLength equ 1024
; Ширина экрана в пикселах
ScreenLength equ 640
; Высота экрана, строк
ScreenHeigth equ 480

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Номер строки "активного" символа
ActiveCharString DW ?
; Номер колонки "активного" символа
ActiveCharColumn DW ?
; Текстовые сообщения
Text DB 10,13,"Выберите символ для просмотра "
     DB "в увеличенном масштабе:",0
     DB 25,14,"Для выбора символа в таблице "
     DB "используйте управляющие",0
     DB 26,14,"клавиши: '",18h,"','",19h
     DB "','",1Ah,"','",1Bh,"'.",0
PEsc DB 29,20,"Для выхода из программы нажмите ESC",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC ShowFont
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; Установить режим прямой адресации памяти
        call    Initialization
; "Захватить" текстовый шрифт
        call    GrabRusFont
; Установить видеорежим
        call    SetVESAVideoMode
```

```
; Отобразить текстовые сообщения
        ; Установить черный цвет фона
        mov     [DefaultBackground],BLACK
        ; Установить зеленый цвет текста
        mov     [DefaultColor],LIGHTGREEN
        MGShowText 3,Text
        ; Установить желтый цвет текста
        mov     [DefaultColor],YELLOW
        MGShowString PEsc
; Инициализировать переменные
        mov     [ActiveCharString],0
        mov     [ActiveCharColumn],0

; ОСНОВНОЙ ЦИКЛ
@@GetCommand:
; Отобразить шрифт
        call    ShowRusFont
; Отобразить укрупненно "активный" символ
        call    ShowLargeChar
; Ввести команду
        call    GetChar
        cmp     AL,0        ;введена команда?
        jne     @@Error
; Проанализировать код команды и выполнить
; соответствующую операцию
        cmp     AH,B_Esc    ;"Выход"
        je      @@End
@@TestBS:
        cmp     AH,B_BS     ;"Стрелка влево"
        jne     @@TestFWD
        ; Проверить значение номера колонки
        ; (минимально допустимое значение - 0)
        cmp     [ActiveCharColumn],0
        je      @@Error
        ; Уменьшить номер колонки
        dec     [ActiveCharColumn]
        jmp short @@GetCommand
@@TestFWD:
        cmp     AH,B_FWD    ;"Стрелка вправо"
        jne     @@TestUp
        ; Проверить значение номера колонки
        ; (максимально допустимое значение - 31)
        cmp     [ActiveCharColumn],31
        jae     @@Error
        ; Увеличить номер колонки
        inc     [ActiveCharColumn]
        jmp short @@GetCommand
@@TestUp:
        cmp     AH,B_UP     ;"Стрелка вверх"
        jne     @@TestDn
```

продолжение ⇒

Листинг 4.13 (*продолжение*)

```
        ; Проверить значение номера строки
        ; (минимально допустимое значение - 0)
        cmp     [ActiveCharString],0
        je      @@Error
        ; Уменьшить номер строки
        dec     [ActiveCharString]
        jmp short @@GetCommand
@@TestDn:
        cmp     AH,B_DN    ;"Стрелка вниз"
        jne     @@Error
        ; Проверить значение номера строки
        ; (максимально допустимое значение - 7)
        cmp     [ActiveCharString],7
        jae     @@Error
        ; Увеличить номер строки
        inc     [ActiveCharString]
        jmp short @@GetCommand
@@Error:
        ; Нврная команда - подать звуковой сигнал
        call    Beep
        jmp     @@GetCommand

; ВЫХОД ИЗ ПРОГРАММЫ
@@End:  ; Установить текстовый режим
        mov     AX,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP ShowFont

;****************************************
;*    ОТОБРАЗИТЬ ШРИФТ В ВИДЕ ТАБЛИЦЫ    *
;* Параметры передаются через переменные *
;* ActiveCharString и ActiveCharColumn   *
;****************************************
PROC ShowRusFont near
        pushad
        mov     SI,offset Font8x16
        mov     EDI,[LinearVideoBuffer]
        add     EDI,LogicalStringLength*200+64+4

        mov     [FontString],0
@@m0:   ; Отобразить очередную строку символов
        mov     [FontColumn],0
@@m1:   ; Отобразить очередной символ
        mov     AH,16   ;число строк (байт) в маске символа
        mov     DL,0    ;цвет фона символа
        mov     DH,7    ;цвет символа
```

```
        mov     CX,[ActiveCharString]
        cmp     [FontString],CX
        jne     @@m2
        mov     CX,[ActiveCharColumn]
        cmp     [FontColumn],CX
        jne     @@m2
        mov     DL,1    ;цвет фона "активного" символа
        mov     DH,15   ;цвет "активного" символа
@@m2:   ; Отобразить строку изображения символа
        mov     AL,[SI] ;загрузить очередной байт маски символа
        mov     CX,8
@@m3:   ; Вывести на экран очередную точку изображения символа
        rol     AL,1
        jc      @@m4
        mov     [byte ptr GS:EDI],DL
        jmp short @@m5
@@m4:   mov     [byte ptr GS:EDI],DH
@@m5:   inc     EDI
        loop    @@m3
        inc     SI
        add     EDI,LogicalStringLength-8
        dec     AH
        jnz     @@m2
        sub     EDI,LogicalStringLength*16-16
        inc     [FontColumn]
        cmp     [FontColumn],32
        jb      @@m1
        add     EDI,LogicalStringLength*24-32*16
        inc     [FontString]
        cmp     [FontString],8
        jb      @@m0
        popad
        ret
ENDP ShowRusFont

;*************************************************
;* ОТОБРАЗИТЬ СИМВОЛ В УВЕЛИЧЕННОМ МАСШТАБЕ (8:1) *
;* Номер отображаемого символа определяется      *
;* переменными ActiveCharString и ActiveCharColumn *
;*************************************************
PROC ShowLargeChar near
        pushad
; Отобразить символ сверху, по центру экрана
        mov     EDI,[LinearVideoBuffer]
        add     EDI,(640-64)/2
; Вычислить положение маски символа в массиве шрифта
        ; Загрузить указатель на шрифт
        mov     SI,offset Font8x16
        ; Умножить номер строки на 32
        mov     AX,[ActiveCharString]
```

продолжение ⇨

Листинг 4.13 (*продолжение*)

```
        shl     AX,5
        ; Прибавить номер колонки
        add     AX,[ActiveCharColumn]
        ; Умножить на размер символа в байтах (на 16)
        shl     AX,4
        add     SI,AX

; Отобразить символ в масштабе 8:1 (размер точки
; символа - 8x8 точек экрана)
        mov     DX,16       ;высота маски символа в точках
@@m1:   mov     AH,8        ;строку символа повторить 8 раз
@@m2:   ;Отображаем строку символа
        mov     AL,[SI]     ;прочитать байт маски
        mov     CX,8        ;ширина маски символа в точках
@@m3:   rol     AL,1
        jc      @@m4
        ;Отобразить 8 точек синего цвета
        mov     [dword ptr GS:EDI],01010101h
        add     EDI,4
        mov     [dword ptr GS:EDI],01010101h
        jmp short @@m5
@@m4:   ;Отобразить 8 точек желтого цвета
        mov     [dword ptr GS:EDI],0E0E0E0Eh
        add     EDI,4
        mov     [dword ptr GS:EDI],0E0E0E0Eh
@@m5:   add     EDI,4
        loop    @@m3
        ; Перейти на следующую строку изображения символа
        add     EDI,LogicalStringLength-64
        dec     AH
        jnz     @@m2
        ; Перейти на следующую строку маски символа
        inc     SI
        dec     DX
        jnz     @@m1
        popad
        ret
ENDP ShowLargeChar
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить набор процедур общего назначения,
; предназначенных для установки графических
; видеорежимов и работы в них
```

```
include "list4_02.inc"
; Подключить набор процедур вывода текста,
; предназначенных для 256-цветных режимов
include "list4_03.inc"

END
```

Анимация двухмерных изображений

Анимация в двухмерном режиме, то есть создание на экране иллюзии движения плоских изображений (спрайтов), также реализуется при помощи масок [1]. Однако если для неподвижного объекта (например, буквы шрифта) достаточно одной маски, то движущийся объект требует отдельную маску для каждой фазы движения в каждом из возможных направлений. Чтобы получить приличную иллюзию движения, необходимо отобразить от 8 до 12 фаз [20]. Человеческий глаз способен различать углы менее одного градуса, поэтому, чтобы создать иллюзию вращения крупного объекта или движения его по произвольным направлениям, нужно иметь по маске на каждый из возможных трехсот шестидесяти градусов поворота.

Сложные объекты способны совершать разнообразные виды движений, для каждого из которых нужен собственный комплект масок. Объект типа «человек», например, может идти, ползти, прыгать, приседать, взбираться по лестнице, плыть и т. п. Механические объекты обычно проще, чем живые, — требуют меньшего количества фаз и разновидностей движения (поэтому их так любят использовать разработчики игр).

Современные компьютерные игры (например, стратегические) предусматривают одновременный вывод на экран нескольких сотен или даже тысяч объектов (подвижных и неподвижных). Можно сформировать группы, считая однотипными те объекты, для отображения которых пригодны одинаковые комплекты масок, однако количество групп все равно измеряется десятками. Одновременно на экран выводится до десяти типов танков, несколько типов самолетов, несколько типов солдат и т. д. В результате общее количество масок, которое необходимо хранить в памяти компьютера, является произведением количества групп объектов, умноженного на количество возможных видов движений объекта, на количество фаз движений и на количество направлений движения.

Рассмотрим следующий пример: пусть имеется 10 типов объектов, способных совершать по два вида движений; при отображении объектов используется 10 фаз, объекты могут двигаться по 50 направлениям. Тогда в общей сложности нужно десять тысяч масок! Каждую

маску необходимо предварительно нарисовать, для чего применяются специальные анимационные программные пакеты (проводить такую работу ручным способом слишком дорого и долго). Кроме того, одна маска сравнительно небольшого объекта размером 32×32 пиксела занимает от 1 до 4 Кбайт памяти, а десять тысяч масок требуют соответственно 10–40 Мбайт!

Как следует из приведенного примера, при разработке компьютерных игр и других программ, использующих анимацию, приходится искусственно ограничивать количество типов объектов, видов и направлений движения.

В общем случае цикл вывода движущегося объекта включает следующую последовательность операций.

1. Вычисление координат области отображения объекта.
2. Сохранение фона области отображения.
3. Рисование текущей фазы движения объекта.
4. Ожидание начала обратного хода луча по кадру.
5. Стирание изображения объекта путем восстановления фона.

Для сохранения фона нужно выделить область памяти такого же размера, что и для маски отображаемой фазы движения. Если маска объекта имеет размер M×N пикселов, а цвет пиксела кодируется K байтами, то область сохранения фона занимает M×N×K байт. При работе с группой движущихся объектов необходимо запоминать последовательность их рисования на экране, поскольку стирание должно проводиться в порядке, обратном порядку вывода.

Вывод изображения на экран обязательно должен быть синхронизирован с началом хода луча по кадру — в противном случае изображение движущегося объекта будет искажаться всякий раз, когда совпадают момент перерисовки и вывода объекта на экран. Если в режиме анимации используется только одна видеостраница, то обязательно необходимо учитывать скорость вывода изображения в видеопамять — даже при наличии синхронизации с ходом луча по кадру и высокой скорости вывода данных в верхней части картинки образуется зона искажений. В зависимости от алгоритма вывода изображений возможны два основных типа искажений — разрезание картинки или непрорисовка некоторых ее участков (рис. 4.15).

Ширина зоны искажений определяется количеством и размером динамических (движущихся или изменяющихся) объектов, их размерами и скоростью вывода изображения на экран. Зона искажений заканчивается в том месте экрана, где при любом возможном количестве и порядке вывода объектов центральный процессор заведомо

опережает луч электронно-лучевой трубки (ЭЛТ). Обычно изготовители компьютерных игр идут на маленькую хитрость — занимают верхнюю часть экрана широким статическим изображением (например, графическим меню).

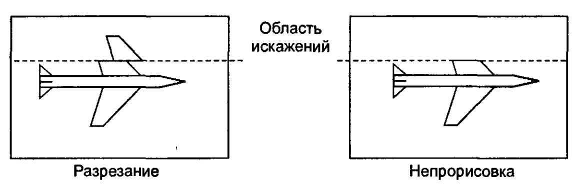


Рис. 4.15. Основные виды искажений при выводе движущихся объектов

При использовании одной видеостраницы движение объектов по экрану будет выглядеть плавным только в том случае, если процессор успевает полностью перерисовывать изображение в течение одного кадра ЭЛТ. Поскольку частота кадров у современных мониторов обычно составляет 65–100 Гц, процессор должен успеть перерисовать кадр за 10–15 миллисекунд, причем для каждого объекта нужно (по изложенному выше алгоритму) успеть:

- восстановить фон в предыдущей позиции;
- сохранить фон в новой позиции;
- нарисовать изображение.

Кроме того, допустимая высота зоны искажений обычно составляет не более 1/5 высоты экрана: пока луч находится в этой зоне, процессор должен успеть перерисовать все подвижные объекты в остальной части экрана.

Проведем оценочные расчеты при условии, что скорость вывода данных составляет 10 млн пикселов в секунду. За время одного кадра (15 мс) процессор успеет перерисовать 150 тыс пикселов, но это число нужно поделить на 3 (для каждого объекта — 3 операции) и на 5 (под зону искажений выделена 1/5 часть экрана). В результате получается, что сумма размеров (в байтах) всех массивов-масок не должна превышать 10 Кбайт. Например, если размер маски равен 1 Кбайт (небольшой объект 32x32 в 256-цветном режиме), то можно вывести 10 движущихся объектов, а при размере маски 4 Кбайт (32x32, режим TrueColor) — всего 2 объекта!

Чтобы ускорить вывод данных, разработчикам программ приходится применять множество разнообразных хитростей. Первая хитрость

основана на том, что современные компьютеры имеют большой объем оперативной памяти, и фон можно в ней хранить целиком, а не кусочками. Ускорение работы при этом получается даже больше, чем на треть, поскольку операция чтения из видеопамяти выполняется медленнее, чем запись в нее (видеоконтроллеры всегда оптимизированы по записи).

Второй трюк заключается в том, чтобы избавиться от операций контроля пересечения границ экрана при выводе маски объекта. Если видеопамять организована обычным образом, а контроля пересечения границ нет, то возможно появление ложных изображений вдоль левой и правой границ экрана (рис. 4.16, *сверху*) или зависание программы при выходе за пределы видеопамяти. Проблема легко решается, если объем видеопамяти позволяет установить логическую длину строки больше физической — при выходе за боковые границы маска попадает в неотображаемую область памяти (рис. 4.16, *снизу*). Кроме того, выравнивание логической длины строки на 2^N пикселов позволяет (хотя и весьма незначительно) упростить расчет координат точек изображения.

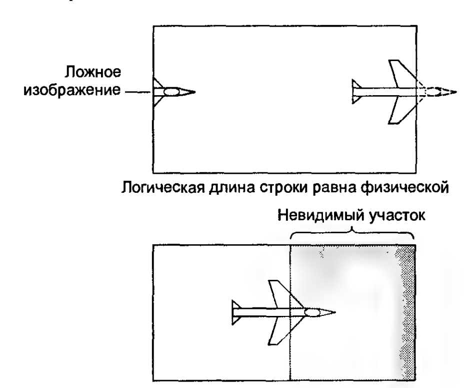


Рис. 4.16. Использование невидимого участка видеопамяти для подавления ложных изображений

Рисунок 4.16 создает впечатление, что ложное изображение подавляется только при выходе за правую границу. На самом деле подавление происходит и в случае выхода за левый край экрана: хорошей

моделью развертки линейной памяти контроллера на экран является спираль (пружина), показанная на рис. 4.17. При выходе за любую боковую границу маска попадает на невидимый участок.

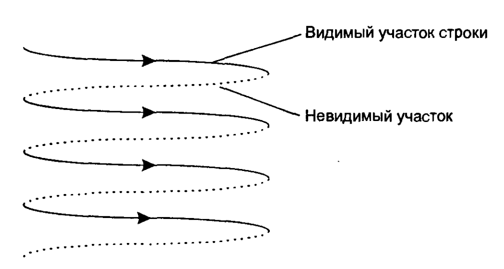


Рис. 4.17. Модель развертки видеопамяти на экран монитора

Можно избавиться от необходимости контролировать пересечение не только боковых, но также верхней и нижней границ. С этой целью создаются горизонтальные неотображаемые области, точнее — горизонтальная область над страницей, поскольку невидимая область под страницей присутствует изначально (см. рис. 4.1 и 4.18). Ширина вертикальной невидимой области должна быть больше или равна максимальной возможной ширине маски L_{max}. Высота горизонтальной области должна быть равна максимальной возможной высоте маски H_{max}.

К сожалению, трюки с защитными полосами применимы не всегда — очень часто по крайней мере одну из сторон изображения занимает полоса меню управления. Проще всего контролировать нижнюю границу спрайта, поэтому (если имеется возможность выбора) лучше всего размещать полосу меню снизу.

Самый эффективный способ устранения искажений анимированного изображения состоит в использовании упомянутого выше механизма переключения страниц: пока видеоконтроллер отображает на экран одну страницу, процессор перерисовывает другую. Однако при этом расход видеопамяти увеличивается вдвое — на старых видеоконтроллерах ее может оказаться недостаточно для создания второй страницы. Кроме того, невидимые области также съедают заметный объем памяти. Организация видеопамяти в режиме переключения страниц при наличии невидимых областей показана на рис. 4.18.

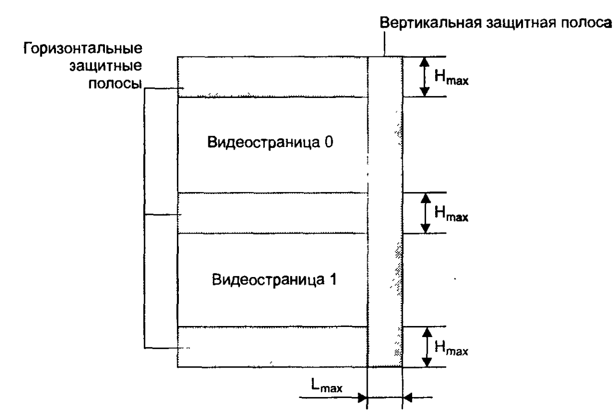


Рис. 4.18. Организация видеопамяти в режиме переключения страниц при наличии защитных неотображаемых полос

При переключении страниц требования к скорости перерисовки кадра значительно ниже, поскольку процессор может заниматься выводом данных в течение всего времени хода луча по кадру. Кроме того, уже не обязательно перерисовывать кадры со скоростью их вывода на экран монитора, то есть переключение страниц можно выполнять не в каждом кадре, а через один или через два (но обязательно — через одинаковое число кадров). Для создания приемлемого изображения достаточно генерировать 25–30 кадров в секунду, то есть на перерисовку каждого кадра можно тратить 30–40 миллисекунд. В 256-цветном режиме с разрешением 640×480 точек за это время можно полностью перерисовать весь экран!

Программа, реализующая анимацию в одностраничном режиме, показана в листингах 4.14–4.20. Листинг 4.14 — это усовершенствованная версия программы из листинга 2.1, реализующая доступ к видеопамяти и дополнительной оперативной памяти через раздельные сегменты большого размера с контролем границ. Пределы сегментов установлены в расчете на *минимальную* конфигурацию современных ПК — 4 Мбайт ОЗУ и 2 Мбайт видеопамяти, но в примере используется только часть (меньше половины) выделенного пространства. В принципе, размеры указанных сегментов можно задавать динамически (в соответствии с конфигурацией конкретного компьютера), однако недавно возникла проблема определения объе-

ма ОЗУ, если он больше 64 Мбайт (предел, на который были когда-то рассчитаны стандартные функции BIOS [85]). Процедура GInitialization использует для загрузки теневых регистров вспомогательную подпрограмму SetSegAddrModeForFSGS, а затем разблокирует линию A20 при помощи подпрограмм Enable_A20 и устанавливает обработчик прерывания по нарушению границ сегментов MemoryProtectionInterrupt при помощи процедуры SetProtectionInterrupt.

Листинг 4.14. Подпрограмма, выполняющая настройку регистра FS на видеопамять, регистра GS — на дополнительную память

```
; Порт, управляющий запретом немаскируемых прерываний
CMOS_ADDR       equ  0070h
CMOS_DATA       equ  0071h
; Селекторы сегментов
SYS_PROT_CS     equ  0008h
SYS_REAL_SEG    equ  0010h
SYS_VIDEO_SEG   equ  0018h
SYS_LINEAR_SEG equ  0020h

DATASEG
; Текстовые сообщения
ProtErr DB LIGHTRED,12,0
        DB "Ошибка: нарушение границ сегмента памяти",0
        DB YELLOW,24,29,"Нажмите любую клавишу",0
; Область сохранения старого вектора прерывания
; по IRQ5 и защите памяти
OldProtInterruptOffset  DW ?
OldProtInterruptSegment DW ?
ENDS

CODESEG
;********************************
;* Подготовка системы к работе *
;********************************
PROC GInitialization NEAR
        pushad
; Занести линейный адрес видеопамяти (для FS)
        mov     EAX,[LinearVideoBuffer]
        mov     [word ptr CS:GDT+26],AX
        ror     EAX,16
        mov     [byte ptr CS:GDT+28],AL
        mov     [byte ptr CS:GDT+31],AH
; Установить предел видеопамяти 2 Мб
        mov     EAX,2*100000h
        shr     EAX,12 ;поделить на гранулярность (4 Кб)
        dec     EAX    ;уменьшить на 1
        mov     [word ptr CS:GDT+24],AX
```

продолжение ⇗

Листинг 4.14 (*продолжение*)

```
        ror     EAX,16
        or      [byte ptr CS:GDT+30],AL
; Занести линейный адрес расширенной памяти (для GS)
        mov     EAX,110000h         ;1 Мб + 64 Кб
        mov     [word ptr CS:GDT+34],AX
        ror     EAX,16
        mov     [byte ptr CS:GDT+36],AL
        mov     [byte ptr CS:GDT+39],AH
; Установить предел расширенной памяти
; (4 Мб минус 1,064 Мб)
        mov     EAX,4*100000h-110000h
        shr     EAX,12 ;поделить на гранулярность (4 Кб)
        dec     EAX    ;уменьшить на 1
        mov     [word ptr CS:GDT+32],AX
        ror     EAX,16
        or      [byte ptr CS:GDT+38],AL
; Сохранить значения сегментных регистров в
; реальном режиме (кроме GS)
        mov     [CS:Save_SP],SP
        mov     AX,SS
        mov     [CS:Save_SS],AX
        mov     AX,DS
        mov     [CS:Save_DS],AX
; (работаем теперь только с кодовым сегментом)
        mov     AX,CS
        mov     [word ptr CS:Self_Mod_CS],AX
        mov     DS,AX
        cli
        mov     SS,AX
        mov     SP,offset Local_Stk_Top
        sti

; Установить режим линейной адресации
        call    SetSegAddrModeForFSGS

; Восстановить значения сегментных регистров
        cli
        mov     SP,[CS:Save_SP]
        mov     AX,[CS:Save_SS]
        mov     SS,AX
        mov     AX,[CS:Save_DS]
        mov     DS,AX
        sti
; Разрешить работу линии A20
        call    Enable_A20
; Замаскировать прерывание IRQ5 и установить
; обработчик прерывания по нарушению границ сегментов
        call    SetProtectionInterrupt
        popad
        ret
```

```
ENDP GInitialization

; Область сохранения значений сегментных регистров
Save_SP DW ?
Save_SS DW ?
Save_DS DW ?
; Указатель на GDT
GDTPtr  DQ ?
; Таблица дескрипторов сегментов для
; входа в защищенный режим
GDT DW 00000h,00000h,00000h,00000h ;не используется
    DW 0FFFFh,00000h,09A00h,00000h ;сегмент кода CS
    DW 0FFFFh,00000h,09200h,00000h ;сегмент данных DS
    DW 00000h,00000h,09200h,000F0h ;сегмент FS
    DW 00000h,00000h,09200h,000F0h ;сегмент GS
; Локальный стек для защищенного режима
; (организован внутри кодового сегмента)
label GDTEnd word
        DB 255 DUP(0FFh)
Local_Stk_Top DB (0FFh)

;*****************************************
;* Процедура, изменяющая содержимое теневых *
;* регистров FS и GS                        *
;*****************************************
PROC SetSegAddrModeForFSGS near
; Вычислить линейный адрес кодового сегмента
        mov     AX,CS
        movzx   EAX,AX
        shl     EAX,4   ;умножить номер параграфа на 16
        mov     EBX,EAX ;сохранить линейный адрес в EBX
; Занести младшее слово линейного адреса в дескрипторы
; сегментов кода и данных
        mov     [word ptr CS:GDT+10],AX
        mov     [word ptr CS:GDT+18],AX
        ; Переставить местами старшее и младшее слова
        ror     EAX,16
; Занести биты 16-23 линейного адреса в дескрипторы
; сегментов кода и данных
        mov     [byte ptr CS:GDT+12],AL
        mov     [byte ptr CS:GDT+20],AL

; Установить предел (Limit) и базу (Base) для GDTR
        add     EBX, offset GDT
        mov [word ptr CS:GDTPtr],(offset GDTEnd-GDT-1)
        mov     [dword ptr CS:GDTPtr+2],EBX
; Сохранить регистр флагов
        pushf
; Запретить прерывания, так как таблица прерываний IDT
; не сформирована для защищенного режима
```

продолжение ⇨

Листинг 4.14 (*продолжение*)

```
        cli
; Запретить немаскируемые прерывания NMI
        in      AL,CMOS_ADDR
        mov     AH,AL
        or      AL,080h       ;установить старший разряд
        out     CMOS_ADDR,AL ;не затрагивая остальные
        and     AH,080h
        ; Запомнить старое состояние маски NMI
        mov     CH,AH
; Перейти в защищенный режим
        lgdt    [fword ptr CS:GDTPtr]
        mov     BX,CS    ;запомнить сегмент кода
        mov     EAX,CR0
        or      AL,01b   ;установить бит PE
        mov     CR0,EAX  ;защита разрешена
        ; Безусловный дальний переход на метку SetPMode
        ; (очистить очередь команд и перезагрузить CS)
                DB      0EAh
                DW      (offset SetPMode)
                DW      SYS_PROT_CS
SetPMode:
        ; Подготовить границы сегментов
        mov     AX,SYS_REAL_SEG
        mov     SS,AX
        mov     DS,AX
        mov     ES,AX
        ; Настроить сегмент FS на видеопамять
        mov     AX,SYS_VIDEO_SEG
        mov     FS,AX
        ; Настроить сегмент GS на расширенную память
        mov     AX,SYS_LINEAR_SEG
        mov     GS,AX
; Вернуться в реальный режим
        mov     EAX,CR0
        and     AL,11111110b ;сбросить бит PE
        mov     CR0,EAX      ;защита отключена
        ; Безусловный дальний переход на метку SetRMode
        ; (очистить очередь команд и перезагрузить CS)
           DB 0EAh
           DW (offset SetRMode)
Self_Mod_CS DW ?

SetRMode:
        ; Регистры стека и данных
        ; настроить на сегмент кода
        mov     SS,BX
        mov     DS,BX
        ; Обнулить ES
```

```
        xor     AX,AX
        mov     ES,AX
        ; Возврат в реальный режим,
        ; прерывания снова разрешены
        in      AL,CMOS_ADDR
        and     AL,07Fh
        or      AL,CH
        out     CMOS_ADDR,AL
        popf
        ret
ENDP SetSegAddrModeForFSGS

;*****************************************
;* Разрешить работу с памятью выше 1 Мб *
;*****************************************
PROC Enable_A20 near
        call    Wait8042BufferEmpty
        mov     AL,0D1h ;команда управления линий A20
        out     64h,AL
        call`   Wait8042BufferEupty
        mov     AL,0DFh ;разрешить работу линии
        out     60h,AL
        call    Wait8042BufferEmpty
        ret
ENDP Enable_A20

;***********************************************
;*   ОЖИДАНИЕ ОЧИСТКИ ВХОДНОГО БУФЕРА I8042    *
;* При выходе из процедуры:                    *
;* флаг ZF установлен - нормальное завершение, *
;* флаг ZF сброшен - ошибка тайм-аута.         *
;***********************************************
proc Wait8042BufferEmpty near
        push    CX
        mov     CX,0FFFFh  ;задать число циклов
@@kb:   in      AL,64h     ;получить статус
        test    AL,10b     ;буфер i8042 свободен?
        loopnz  @@kb       ;если нет, то цикл
        pop     CX
        ; (если при выходе сброшен флаг ZF - ошибка)
        ret
endp Wait8042BufferEupty

;**********************************
;* УСТАНОВИТЬ ВЕКТОР ПРЕРЫВАНИЯ ПО *
;*   СРАБАТЫВАНИЮ ЗАЩИТЫ ПАМЯТИ    *
;**********************************
PROC SetProtectionInterrupt NEAR
        pusha
        push    ES
```

продолжение ⇨

Листинг 4.14 (*продолжение*)

```
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
; Замаскировать прерывание IRQ5
        cli
        in      AL,21h
        or      AL,100000b
        out     21h,AL
        sti
; Установить вектор прерывания
        ; Настроить ES на область векторов
        mov     AX,0
        mov     ES,AX
        ; Сохранить старый вектор
        mov     AX,[ES:13*4]
        mov     [OldProtInterruptOffset],AX
        mov     AX,[ES:13*4+2]
        mov     [OldProtInterruptSegment],AX
        ; Установить новый вектор
        cli
        mov     AX,offset MemoryProtectionInterrupt
        mov     [ES:13*4],AX
        mov     AX,CS
        mov     [ES:13*4+2],AX
        sti
        pop     ES
        popa
        ret
ENDP SetProtectionInterrupt

;*************************************
;* ВОССТАНОВИТЬ СТАРЫЙ ВЕКТОР ЗАЩИТЫ *
;*************************************
PROC RestoreOldProtectionInterrupt NEAR
        pusha
        push    ES
        ; Настроить ES на область векторов
        mov     AX,0
        mov     ES,AX
        ; Восстановить старый вектор
        cli
        mov     AX,[OldProtInterruptOffset]
        mov     [ES:13*4],AX
        mov     AX,[OldProtInterruptSegment]
        mov     [ES:13*4+2],AX
        sti
        pop     ES
```

```
        popa
        ret
ENDP RestoreOldProtectionInterrupt

;***********************************
;*   НОВЫЙ ОБРАБОТЧИК ПРЕРЫВАНИЯ ПО  *
;* НАРУШЕНИЮ ГРАНИЦ СЕГМЕНТОВ ПАМЯТИ *
;***********************************
PROC MemoryProtectionInterrupt NEAR
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 2,ProtErr
; Ожидать нажатия клавиши
        call    GetChar
; Восстановить старый обработчик прерывания
        call    RestoreOldProtectionInterrupt
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP MemoryProtectionInterrupt
ENDS
```

После выполнения процедуры GInitialization к видеопамяти можно обращаться через сегментный регистр FS, а к дополнительной оперативной памяти — через GS. В случае нарушения установленных границ сегментов происходит прерывание общей защиты памяти, и процедура MemoryProtectionInterrupt выдает сообщение об ошибке, а затем осуществляет аварийное завершение работы программы с немедленным выходом в DOS.

ВНИМАНИЕ

Система защиты памяти и параллельный порт LPT2 настроены на один и тот же вектор, поэтому сигнал от LPT2 обязательно нужно маскировать при установке обработчика сигнала защиты памяти.

Листинг 4.15 является видоизмененным для сегментной адресации вариантом листинга 4.3. Он включает две подпрограммы: процедуру вывода символа PutGraChar и процедуру очистки экрана GClearScreen для 256-цветных режимов с линейным видеобуфером.

Листинг 4.15. Процедуры вывода символа и очистки экрана для 256-цветных режимов с линейной адресацией видеобуфера

```
DATASEG
; Цвет текста в графическом режиме по умолчанию
DefaultColor      DB WHITE  ;белый
; Цвет фона в графическом режиме по умолчанию
DefaultBackground DB BLACK  ;черный
ENDS

CODESEG
;*****************************************************
;* ВЫВОД СИМВОЛА 8x16 НА ЭКРАН В ГРАФИЧЕСКОМ РЕЖИМЕ *
;*            (для 256-цветных режимов)              *
;* Все параметры передаются через регистры:          *
;* AL - ASCII-код символа;                           *
;* DH - номер текстовой строки экрана;               *
;* DL - номер текстовой колонки экрана;              *
;* Используются цвет символов и цвет фона,           *
;* заданные по умолчанию.                            *
;*****************************************************
PROC PutGraChar NEAR
        push    DS
        pushad
        mov     CX,[CS:MainDataSeg]
        mov     DS,CX
        cld

        ; Смещение символа от начала шрифта
        mov     SI,offset Font8x16
        xor     AH,AH
        shl     AX,4
        add     SI,AX

        ; Вычислить левый верхний угол символа
        xor     EBX,EBX
        mov     BL,DH
        shl     EBX,14 ;умножить номер строки на 16*1024
        xor     DH,DH
        shl     DX,3   ;умножить номер столбца на 8
        or      BX,DX
        mov     EDI,EBX
        mov     BL,[DefaultColor]
        mov     DL,[DefaultBackground]

        mov     AH,16 ;счетчик строк маски буквы
@@M0:   lodsb
        mov     CX,8  ;счетчик точек в строке маски
@@M1:   rol     AL,1
        jc      @@M2
```

```
        mov     [FS:EDI],DL
        jmp     @@M3
@@M2:   mov     [FS:EDI],BL
@@M3:   inc     EDI
        loop    @@M1
        add     EDI,LogicalStringLength-B
        dec     AH
        jnz     @@M0
;Завершение процедуры
@@EndPutGraChar:
        popad
        pop     DS
        inc     DL
        ret
ENDP PutGraChar

;*************************************
;* ОЧИСТКА ЭКРАНА В ГРАФИЧЕСКОМ РЕЖИМЕ *
;*   (процедура параметров не имеет)   *
;*************************************
PROC GClearScreen NEAR
        pushad
; Умножить высоту экрана ScreenHeigth на логическую
; ширину строки (1024 пиксела)
        mov     ECX,ScreenHeigth
        shl     ECX,10
        mov     EDI,0
; Заполнить видеопамять нулями
        mov     AL,0        ;черный цвет
@@NextPixels:
        mov     [FS:EDI],AL
        inc     EDI
        dec     ECX
        jnz     @@NextPixels
        popad
        ret
ENDP GClearScreen
ENDS
```

В листинге 4.16 приведено описание констант, глобальных переменных и макрокоманд, используемых в листингах 4.18–4.22. Макрокоманды в данном случае делают текст значительно компактнее и практически не маскируют особенности работы программы. Макрос DrawMImage осуществляет подготовку параметров и вызов функции рисования движущегося изображения DrawMovingImage, макрос DeleteMImage — подготовку параметров и вызов функции стирания движущегося изображения DeleteImage, а макрос DrawSImage — подготовку параметров и вызов функции рисования статического изображения DrawStaticImage.

Объем данных в приведенном примере невелик, и потому все массивы размещены в основной области памяти DOS. В программах с большим размером структур данных можно свободно использовать для хранения переменных и массивов расширенную память, однако следует учитывать, что вся ответственность за распределение расширенной памяти при этом *возлагается на программиста*. Обычно количество простых переменных не превышает нескольких тысяч, и для их хранения лучше все-таки применять обычную память, автоматически распределяемую программой-компилятором. В расширенной памяти удобно хранить большие объекты (например, статический фон для экранного изображения) и группы объектов с повторяющейся структурой (например, маски различных фаз движения в программах с двумерной анимацией). При необходимости также можно из специальных файлов загружать прямо в расширенную память крупные текстовые сообщения (используемые обычно в режиме подсказки).

Листинг 4.16. Описание констант, глобальных переменных и макрокоманд

```
; КОНСТАНТЫ
; Режим с разрешением 640x480, 256 цветов
GraphicsMode equ 4101h
; Логическая ширина строки в пикселах
LogicalStringLength equ 1024
; Ширина экрана в пикселах
ScreenLength equ 640
; Высота экрана, строк
ScreenHeigth equ 480
; Количество "облаков" на небе (облака накладываются
; друг на друга)
MaxCloudNum equ 10
; Число кадров взлета (разгона) ракеты
MaxRktStartFrame equ 56
; Число кадров взрыва
MaxExplFrameNumber equ 10
; Число осколков при взрыве
SplinterMaxNumber equ 40
; Радиус "разлета осколков" при взрыве
ExpR equ 32
; Квадрат расстояния "подрыва" (от центра ракеты
; до цента цели)
TargetDistanceSQ equ 576  ;(расстояние подрыва 24)
; Максимально допустимое количество
; пропущенных самолетов
MaxSavedPlanes equ 3
; Максимальное число эпизодов игры
```

```
MaxEpNumber equ 100

; МАКРОСЫ
; Вызов функции рисования движущегося объекта
MACRO DrawMImage ObjS,ObjC,ObjL,ObjH,Obj,ObjA,ObjF
        ; Передать параметры подпрограмме рисования
        mov     AL,[ObjF]
        mov     [ImageF],AL
        mov     EAX,[ObjS]
        mov     [ImageS],EAX
        mov     EAX,[ObjC]
        mov     [ImageC],EAX
        mov     [dword ptr ImageL],ObjL
        mov     [dword ptr ImageH],ObjH
        mov     [ImageMaskOffset], offset Obj
        ; Нарисовать объект
        call    DrawMovingImage
        ; Запомнить адрес изображения в видеопамяти
        mov     EAX,[ImageA]
        mov     [ObjA],EAX
ENDM

; Вызов функции восстановления фона
; (стирание изображения объекта)
MACRO DeleteMImage ObjA,ObjL,ObjH,ObjF
        mov     EAX,[ObjA]
        mov     [ImageA],EAX
        mov     [dword ptr ImageL],ObjL
        mov     [dword ptr ImageH],ObjH
        mov     AL,[ObjF]
        mov     [ImageF],AL
        call    DeleteImage
ENDM

; Вызов функции рисования неподвижного объекта
MACRO DrawSImage ObjS,ObjC,ObjL,ObjH,Obj
        ; Передать параметры подпрограмме рисования
        mov     EAX,[ObjS]
        mov     [ImageS],EAX
        mov     EAX,[ObjC]
        mov     [ImageC],EAX
        mov     [dword ptr ImageL],ObjL
        mov     [dword ptr ImageH],ObjH
        mov     [ImageMaskOffset], offset Obj
        ; Нарисовать объект
        call    DrawStaticImage
ENDM

DATASEG
; ОСНОВНЫЕ ПЕРЕМЕННЫЕ СОСТОЯНИЯ ИГРЫ
; Счетчик игровых эпизодов
```

продолжение ⇨

Листинг 4.16 (*продолжение*)

```
EpisodeNumber DW ?
; Счетчик игрового времени (число видеокадров
; от начала игрового эпизода)
GameTimeCounter DW ?
; Флаг изменения состояния игры
GameStateGhange DB ?
; Состояние самолета (0 - ожидание, 1 - полет,
; 2 - взрыв)
PlnState    DB ?
; Состояние ракеты (0 - ожидание запуска, 1 - запуск,
; 2 - полет, 3 - взрыв, 4 - разлет осколков,
; 5 - ракеты нет на экране)
RktState    DB ?
; Счетчик самолетов
PlnCounter  DW ?
; Счетчик сбитых самолетов
DestroedPlns DW ?
; Счетчик пропущенных самолетов
EscPlns     DW ?
; Счетчик потраченных ракет
RktCounter  DW ?

; ИНФОРМАЦИЯ ОБ ОТОБРАЖАЕМЫХ ОБЪЕКТАХ
; Позиция маски самолета на экране
PlnS   DD ?  ;строка
PlnC   DD ?  ;колонка
PlnA   DD ?  ;текущий линейный адрес
PlnFl  DB ?  ;флаг наличия объекта в текущей странице
; Признак попадания ракеты
HitFlag DW ?
; Скорость движения самолета
PlaneSpeed DD ?
; Направление движения самолета
PlaneDirection DB ?
; Задержка пуска самолета
PlaneDeltaT DW ?
; Позиция маски ракеты на экране
RktS   DD ?  ;строка
RktC   DD ?  ;колонка
RktA   DD ?  ;текущий линейный адрес
RktFl  DB ?  ;флаг наличия объекта в текущей странице
RktStartFrameNumber DW ? ;номер кадра старта ракеты
; Позиция маски пламени на экране
FlmS   DD ?  ;строка
FlmA   DD ?  ;текущий линейный адрес
FlmFl  DB ?  ;флаг наличия объекта в текущей странице
; Параметры взрыва
ExpS   DD ?  ;строка
ExpC   DD ?  ;колонка
```

```
ExpA   DD ?  ;текущий линейный адрес
ExpFl  DB ?  ;флаг наличия объекта в текущей странице
ExpFrameNumber DW ? ;номер кадра взрыва
; Параметры дерева
TreeS     DD ?  ;строка
TreeC     DD ?  ;колонка
; Параметры облака
CloudS    DD ?  ;строка
CloudC    DD ?  ;колонка
; Позиция пусковой установки на экране
PRS       DD ?  ;строка
PRC       DD ?  ;колонка

; ТЕКСТОВЫЕ СООБЩЕНИЯ
EndTxt DB 0,32,"РЕЗУЛЬТАТЫ ИГРЫ",0
       DB 12,0,"Сыграно эпизодов:",0
       DB 14,0,"Выпущено ракет:",0
       DB 16,0,"Сбито самолетов:",0
       DB 18,0,"Пропущено самолетов:",0
       DB 29,29,"Нажмите любую клавишу",0

; ДИНАМИЧЕСКИЕ МАСКИ
PlnMask DB 1024 DUP(?)   ;текущая маска самолета
ExpMask DB 4096 DUP(?)   ;динамическая маска взрыва
ENDS
```

Листинг 4.17 включает в себя маски статических (дерево, облако, пусковая установка) и динамических (самолет, ракета, пламя из сопла ракеты) объектов, а также массивы координат статических объектов и массивы случайных параметров для динамических объектов. К сожалению, в процессе окончательного оформления листингов при подготовке книги к печати приходится придерживаться жестких ограничении на длину строки программы. Каждая строка данных в масках широких объектов (самолета и облака) оказалась поделенной на две части, в результате чего была утрачена наглядность, однако все остальные объекты просматриваются достаточно четко.

Листинг 4.17. Набор масок объектов, массивов координат и случайных чисел для игры «Самолет и ракета»

```
DATASEG
; Маска самолета
PlnML equ 32  ; ширина маски самолета
PlnMH equ 32  ; высота маски самолета
Plane DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
      DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
      DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
      DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
      DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

продолжение ⇨

Листинг 4.17 (*продолжение*)

```
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,8,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,8,7,8,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,8,7,7,8,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,8,7,7,7,8,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,8,7,7,7,8,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 8,8,0,0,0,0,0,0,0,0,8,7,7,7,7,8
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 8,7,8,0,0,0,0,0,0,0,8,7,7,7,7,7
DB 8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 8,7,7,8,0,0,0,0,0,0,0,8,7,7,7,7
DB 7,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 8,7,7,8,0,0,0,0,0,0,0,0,7,7,7,7
DB 7,7,8,0,0,8,8,8,0,0,0,0,0,0,0,0
DB 8,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8
DB 8,8,8,8,8,7,7,7,8,8,8,0,0,0,0,0
DB 0,8,8,8,8,7,7,7,7,7,7,7,7,7,7,7
DB 7,7,7,7,8,8,8,8,8,7,7,8,8,8,0,0
DB 0,8,8,8,8,7,7,7,7,7,7,7,8,8,8,8
DB 8,8,8,7,7,7,7,7,7,7,7,7,7,7,8,8
DB 0,7,7,7,8,7,7,7,7,7,7,8,7,7,7,7
DB 7,7,7,8,7,7,7,7,7,7,7,8,8,8,0,0
DB 8,7,7,B,8,8,8,8,8,8,8,8,7,7,7,7
DB 7,7,8,8,8,8,8,8,8,8,8,0,0,0,0,0
DB 8,7,8,0,0,0,0,0,0,0,0,8,7,7,7,7
DB 7,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 8,8,0,0,0,0,0,0,0,0,8,7,7,7,7,7
DB 8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 8,0,0,0,0,0,0,0,0,0,8,7,7,7,7,8
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,8,7,7,7,8,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,8,7,7,7,8,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,8,7,7,8,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,8,7,8,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,8,0,0,0,0,0
DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

```
        DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DB 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

; Маска ракеты
RktML equ 16 ; ширина маски ракеты
RktMH equ 32 ; высота маски ракеты
Rkt DB  0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 7,15, 7, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 7,15, 7, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 7,15, 7, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 7,15,15,15, 7, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 7,15,15,15, 7, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 7,15,15,15, 7, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 6,10, 2, 6, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 2, 2, 6, 2, 2, 2, 2, 0, 0, 0, 0, 0
    DB  0, 0, 0, 2, 2, 2,10, 2,14, 2, 2, 2, 0, 0, 0, 0
    DB  0, 0, 2, 6, 2, 2, 2, 2, 2, 2,14, 2, 2, 0, 0, 0
    DB  0, 2, 6,14, 6, 2, 6, 2, 6, 2, 6,10, 6, 2, 0, 0
    DB  2, 2, 2, 2, 2, 2, 2, 2,10, 2, 2, 2, 2, 2, 2, 0
    DB  0, 0, 0, 0, 0, 2, 6, 2,14, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2, 6,10, 6, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2,14, 6, 2, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2, 6, 2,10, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2,10,14, 6, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2, 6, 2, 6, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2,14, 6, 2, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2, 6, 2,14, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2,10, 6,10, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 2,14, 2, 2, 2, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 2, 2,10, 2, 6, 2, 2, 0, 0, 0, 0, 0
    DB  0, 0, 0, 2, 6, 2, 2, 2,14, 2,14, 2, 0, 0, 0, 0
    DB  0, 0, 2,14, 2, 2, 6, 2,10, 2, 6, 2, 2, 0, 0, 0
    DB  0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0

; Маска пламени из сопла ракеты
FlmML equ 16  ; ширина маски пламени
FlmMH equ 16  ; высота маски пламени
```

продолжение

Листинг 4.17 (*продолжение*)

```
Flm DB  0, 0, 0, 0, 0,12,15,15,15,12, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0,12,15,15,15,12, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0,12,14,15,14,12, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0,12,14,14,14,12, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0,12,14,14,14,12, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0,12,14,14,14,12, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 4,12,14,12, 4, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 4,12,14,12, 4, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 4,12,14,12, 4, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 4,12,14,12, 4, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 4,12, 4, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 4,12, 4, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 4,12, 4, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0

; Маска дерева
TreeML equ 16   ; ширина маски дерева
TreeMH equ 32   ; высота маски дерева
Tree DB  0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0,10,10,10, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0,10,10,10,10,10, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0,10,10,10, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0,10,10,10,10,10,10,10, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0,10,10,10,10,10, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0,10,10,10,10,10,10,10,10,10, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0,10,10,10,10,10, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0,10,10,10,10,10,10,10,10,10, 0, 0, 0, 0
     DB  0,10,10,10,10,10,10,10,10,10,10,10,10,10, 0, 0
     DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0,10,10,10,10,10,10,10, 0, 0, 0, 0, 0
     DB  0, 0,10,10,10,10,10,10,10,10,10,10,10, 0, 0, 0
     DB 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 0
     DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
     DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
```

```
    DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0
    DB  0, 0, 0, 0, 0, 0, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0

; Маска пусковой установки
PRML equ 16  ; ширина маски пусковой установки
PRMH equ 32  ; высота маски пусковой установки
Platf DB 16,16,16,16,16,16,16,22,16,16,16,16,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7, 2, 2, 2,16,22,16, 2, 2, 2, 7, 7, 7, 0
      DB 16,16,16, 2, 2, 2,16,22,16, 2, 2, 2,16,16,16, 0
      DB  7, 7, 7,16,16,16,16,16,16,16,16,16, 7, 7, 7, 0

; Маска облака
CloML equ 32  ; ширина маски самолета
CloMH equ 16  ; высота маски самолета
Cloud DB  0, 0, 0, 0, 0, 0,2B, 0, 0, 0, 0, 0,31, 0,31,30
      DB 31,31, 0, 0, 0,25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
      DB  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,30, 0,30,30, 0
      DB 30,30,30,30, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
      DB  0, 0, 0,27,32, 0, 0,29,30, 0,2B,29,30,29, 0,29
      DB  0,29,27, 0,30, 0,27, 0, 0, 0,25, 0, 0, 0, 0, 0
      DB  0,24, 0, 0, 0, 0,30, 0,29, 0,31, 0,27,31,29,30
```

продолжение ⇗

Листинг 4.17 (*продолжение*)

```
        DB 31,26, 0,29, 0,28, 0,31, 0,30, 0, 0,29,24, 0, 0
        DB  0, 0,30, 0,27, 0,31,30, 0,29,30,27, 0,29, 0,28
        DB  0,31,27,29,28,30,29,27,26,29, 0,28, 0, 0, 0, 0
        DB  0,23, 0,30, 0,26,31,28,29,27,29, 0,28, 0,29,30
        DB 29,29, 0,27,29, 0,25,29,28,28,26,27,29, 0,25, 0
        DB  0, 0, 0,24,28,29, 0,24,27, 0,26,29,25,24,27, 0
        DB 29,23,30, 0,26,27,24, 0,28, 0,25,30, 0,24, 0, 0
        DB  0,25,29,24, 0,30,26,24,28,27,25, 0,26, 0,26,24
        DB 28,24,25,27,28, 0,25,23,26,27,23,27,24, 0,25,22
        DB 24,23, 0,25,22,27,24, 0,25,23,22,28,24,26,30,23
        DB 25, 0,24,22, 0,25,23,22, 0,25,25, 0,26,24,27, 0
        DB 25, 0,31,23,26, 0,29,24,23, 0,26,24,27,26,23, 0
        DB 22,25,26, 0,29,26,24, 0,26, 0,24,26, 0,25, 0, 0
        DB  0,25,20, 0,23,25,22,25, 0,23, 0,30, 0,24,25,27
        DB 24, 0,27,19,28,26, 0,20,24,22,21,25,23,25,22, 0
        DB 22, 0,23,24,20, 0,24,23,20,21,22,24,25,23, 0,24
        DB 21,22,19, 0,24,23,24,22,25, 0,21, 0,23,20, 0, 0
        DB  0,19,20,24, 0,20,23, 0,18,19,17, 0,21,20,19,20
        DB 18,21,22,20,23, 0,20,23,22,20,23,22,18, 0, 0,19
        DB  0, 0,18, 0,20, 0,17,21,20, 0,19,17,16,19, 0,18
        DB  0,24,18,19, 0,17,20,22,18,17, 0,20, 0,23, 0, 0
        DB  0, 0, 0, 0,20,23, 0,24,20,18,21,23, 0,20,19, 0
        DB 22,21, 0,22,18,19,24, 0,20, 0,19,22,20, 0, 0, 0
        DB  0, 0, 0,18, 0,17,20,19,20,19,21, 0,20,17,19,21
        DB 20,18,17,20,20,19,20,17,20, 0,19,22, 0, 0, 0, 0
        DB  0,22, 0, 0,19, 0, 0,16,17, 0,21,23,25, 0,19, 0
        DB  0,16, 0,17,18, 0, 0, 0,20, 0,19, 0, 0,18, 0, 0

; Массив координат облаков: первое значение в паре -
; номер строки (Y), второе - номер колонки (X)
CloudPos DD 100,200, 200,300, 200,310, 200,320, 160,500
         DD 160,515, 165,505, 300,110, 295,105, 293,115

; Массив пар приращений координат для ракеты и пламени
; из ее сопла: первый элемент пары - приращение
; координаты ракеты, второй - смещение маски
; пламени относительно начала маски ракеты
; (изменяется от 16 до 32).
RktStartStep DD 0,16, 0,17, 0,18, 0,19
             DD 0,20, 0,21, 0,22, 0,23
             DD 0,24, 0,25, 0,26, 0,27
             DD 1,28, 1,29, 2,30, 2,31
             DD 3,32, 3,32, 4,32, 4,32
             DD 5,32, 5,32, 6,32, 6,32
             DD 7,32, 7,32, 8,32, 8,32
             DD 9,32, 10,32, 11,32, 12,32
             DD 13,32, 14,32, 15,32, 16,32
             DD 17,32, 18,32, 19,32, 20,32
             DD 21,32, 22,32, 23,32, 24,32
```

```
            DD 26,32, 28,32, 30,32, 32,32
            DD 34,32, 36,32, 38,32, 40,32
            DD 42,32, 44,32, 46,32, 48,32

; Массив шагов (смещений) осколков из
; 2*SplinterMaxNumber элементов: первый элемент
; пары - шаг по Y, второй - шаг по X
SplStep DB 0,1, 1,0, 0,-1, -1,0, 1,1, 1,-1, -1,1, -1,-1
        DB 0,2, 2,0, 0,-2, -2,0, 2,2, 2,-2, -2,2, -2,-2
        DB 1,2, 1,-2, -1,2, -1,-2, 2,1, 2,-1, -2,1,-2,-1
        DB 1,3, 1,-3, -1,3, -1,-3, 3,1, 3,-1, -3,1,-3,-1
        DB 2,3, 2,-3, -2,3, -2,-3, 3,2, 3,-2, -3,2,-3,-2

; Массив цветов осколков
ExpColors DB WHITE,WHITE,YELLOW,YELLOW,LIGHTRED
          DB LIGHTRED,RED,RED,DARKGREY,DARKGREY

; МАССИВЫ СЛУЧАЙНЫХ ЧИСЕЛ
; Массив смещений самолета по высоте
DeltaS DD 182,3,52,230,162,6,142,227,122,189
       DD 72,91,128,214,209,76,45,169,31,3
       DD 125,153,139,168,27,220,66,176,174,233
       DD 205,83,137,66,184,51,134,192,81,24
       DD 71,185,173,164,69,8,90,233,16,153
       DD 25,69,170,233,79,73,110,15,147,21
       DD 225,81,13,145,161,125,166,167,80,220
       DD 105,228,129,181,162,118,127,207,30,16
       DD 144,84,9,125,146,135,80,1,90,229
       DD 189,235,97,47,193,107,162,30,234,82
; Массив начальных задержек запуска самолета
DeltaT DW 148,83,65,149,54,18,131,2,55,166
       DW 87,97,128,51,117,188,96,163,61,177
       DW 115,197,78,177,56,73,152,144,99,48
       DW 5,35,163,97,96,117,79,70,55,192
       DW 143,11,42,160,141,135,192,83,98,63
       DW 4,183,41,189,63,118,9,2,146,181
       DW 26,31,91,174,58,23,94,182,68,34
       DW 199,14,189,44,63,131,135,172,150,28
       DW 76,102,25,94,76,191,127,104,134,82
       DW 82,189,136,88,121,18,139,162,82,43
; Массив направлений полета (0 - слева направо,
; 1 - справа налево)
Direction DB 0,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,1,0,1,0
          DB 0,0,1,1,0,1,1,0,1,0,1,1,0,0,0,1,1,1,0,0
          DB 0,0,0,1,0,1,0,1,1,1,0,1,1,0,1,0,0,1,1,0
          DB 1,1,1,0,1,1,0,1,0,0,1,0,1,1,1,1,0,0,1,0
          DB 1,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,0,0,0,1
; Массив скоростей полета самолета
PlnSpeed DD 2,1,2,1,1,1,2,1,2,1,2,2,1,2,2,2,2,2,1,2
         DD 2,1,1,2,2,1,2,2,2,1,1,2,1,2,1,2,2,2,2,1
```

продолжение ⇨

Листинг 4.17 (*продолжение*)

```
        DD 2,1,1,2,1,2,1,1,2,2,2,2,2,1,1,1,1,1,2,2
        DD 1,2,2,2,1,1,1,2,2,1,1,1,1,2,2,1,1,2,2,2
        DD 2,1,1,1,1,1,1,2,1,2,2,2,1,2,2,1,1,2,1,2
ENDS
```

ПРИМЕЧАНИЕ

При создании масок я не использовал никаких специальных редакторов, а непосредственно набирал числовые коды цветов из стандартной палитры VGA. Первые 16 кодов стандартной палитры соответствуют кодам 16-цветных режимов (см. листинг 1.5), а следующие 16 кодов — оттенкам серого цвета (от черного до белого). Используя анимационные пакеты или самодельные редакторы изображений, можно создавать гораздо более красивые маски, если задействовать все 256 доступных оттенков и перепрограммировать регистры ЦАП при помощи функции 10h прерывания Int 10h (подфункция 12h).

Листинг 4.18 содержит группу подпрограмм, специфических для одностраничного режима (для режима с переключением страниц в них нужно вносить дополнения). Процедура DrawMovingImage обеспечивает вывод динамических (движущихся) изображений в видеопамять, а DeleteImage — их стирание путем восстановления фона. Для рисования статических объектов фона используется подпрограмма DrawStaticImage. Процедура ShowBackground полностью копирует фон в видеопамять в начале каждого нового эпизода. Процедура ShowEscapedPlanes служит для вывода в левый верхний угол экрана количества пропущенных самолетов.

Листинг 4.18. Набор подпрограмм для вывода фона и динамических объектов в одностраничном режиме

```
; Номер начальной строки видеостраницы
VPage0StartString equ 0
; Начальный адрес видеостраницы
VPage0Address     equ 0

DATASEG
; ПАРАМЕТРЫ ПОДПРОГРАММЫ РИСОВАНИЯ СПРАЙТА
; Указатель на маску объекта
ImageMaskOffset DW ?
; Размеры маски изображения
ImageL DD ? ;ширина маски
ImageH DD ? ;высота маски
; Позиция маски изображения на экране
ImageS DD ? ;строка
ImageC DD ? ;колонка
ImageA DD ? ;линейный адрес
```

```
ImageF DB ? ;флаг присутствия объекта на экране
ENDS

CODESEG
;*****************************************************
;* ПЕРЕПИСАТЬ ФОН ИЗ ОПЕРАТИВНОЙ ПАМЯТИ В ВИДЕОПАМЯТЬ *
;*          (процедура параметров не имеет)          *
;*****************************************************
PROC ShowBackground near
        pushad
        cld
        mov     EDI,0
        ; Загрузить в счетчик размер области фона
        mov     ECX,LogicalStringLength*ScreenHeigth
        ; Скопировать фон
@@N:    mov     AL,[GS:EDI]
        mov     [FS:EDI],AL
        inc     EDI
        dec     ECX
        jnz     @@N
        popad
        ret
ENDP ShowBackground

;************************************************
;*          НАРИСОВАТЬ ДИНАМИЧЕСКИЙ ОБЪЕКТ        *
;* Для передачи параметров используются глобальные *
;* переменные ImageMaskOffset, ImageL, ImageH,    *
;* ImageS, ImageC, ImageA, ImageF.                *
;************************************************
PROC DrawMovingImage near
        pushad
        cmp     [ImageF],0
        je      @@End
        cld
; Вычислить адрес начальной точки для вывода маски
        ; Умножить длину строки на номер строки (Y)
        mov     EAX,LogicalStringLength
        mov     EDX,[ImageS]
        mul     EDX
        ; Прибавить номер колонки (X)
        add     EAX,[ImageC]
        mov     [ImageA],EAX ;запомнить смещение
        mov     EDI,EAX ;результат - в индексный регистр
        ; Записать адрес маски в индексный регистр
        mov     SI,[ImageMaskOffset]
; Вывести изображение
        mov     DX,[word ptr ImageH] ;высота маски
@@M0:   ; Вывести очередную строку маски
        mov     CX,[word ptr ImageL] ;ширина маски
```

продолжение ⇨

Листинг 4.18 (*продолжение*)

```
@@M1:   ; Проверить точку маски
        lodsb
        and     AL,AL           ;код цвета равен нулю?
        jz      @@M2            ;пропустить точку
        mov     [FS:EDI],AL     ;вывести точку
@@M2:   ; Перейти к следующей точке
        inc     EDI
        loop    @@M1
        ; Перейти на следующую строку
        add     EDI,LogicalStringLength
        sub     EDI,[ImageL]
        dec     DX
        jnz     @@M0
@@End:  popad
        ret
ENDP DrawMovingImage

;****************************************************
;*              НАРИСОВАТЬ ЭЛЕМЕНТ ФОНА             *
;* Для передачи параметров используются глобальные *
;* переменные ImageMaskOffset, ImageL, ImageH,      *
;* ImageS, ImageC.                                  *
;****************************************************
PROC DrawStaticImage near
        pushad
        cld
; Загрузить начальное смещение
        mov     EDI,VPage0Address
; Вычислить адрес начальной точки для вывода маски
        ; Умножить длину строки на номер строки (Y)
        mov     EAX,LogicalStringLength
        mov     EDX,[ImageS]
        mul     EDX
        ; Прибавить номер колонки (X)
        add     EAX,[ImageC]
        ; Прибавить результат к начальному смещению
        add     EDI,EAX
        ; Записать адрес маски в индексный регистр
        mov     SI,[ImageMaskDffset]
; Вывести изображение
        mov     DX,[word ptr ImageH] ;высота маски
@@M0:   ; Вывести очередную строку маски
        mov     CX,[word ptr ImageL] ;ширина маски
@@M1:   ; Проверить точку маски
        lodsb
        and     AL,AL           ;код цвета равен нулю?
        jz      @@M2            ;пропустить точку
        mov     [GS:EDI],AL     ;вывести точку
@@M2:   ; Перейти к следующей точке
```

```
        inc     EDI
        loop    @@M1
        ; Перейти на следующую строку
        add     EDI,LogicalStringLength
        sub     EDI,[ImageL]
        dec     DX
        jnz     @@M0
        popad
        ret
ENDP DrawStaticImage

;************************************************
;*      СТЕРЕТЬ ИЗОБРАЖЕНИЕ (ВОССТАНОВИТЬ ФОН)    *
;* Для передачи параметров используются глобальные *
;* переменные ImageL, ImageH, ImageA, ImageF.      *
;************************************************
PROC DeleteImage near
        pushad
        cmp     [ImageF],0
        je      @@End
; Записать в индексные регистры адрес изображения
        mov     EDI,[ImageA]
; Вывести исходное изображение
        mov     DX,[word ptr ImageH] ;высота маски
@@M0:   mov     CX,[word ptr ImageL] ;ширина маски
@@M1:   mov     AL,[GS:EDI]
        mov     [FS:EDI],AL
        inc     EDI
        loop    @@M1
        add     EDI,LogicalStringLength
        sub     EDI,[ImageL]
        dec     DX
        jnz     @@M0
@@End:  popad
        ret
ENDP DeleteImage

;*****************************************
;* ОТОБРАЗИТЬ ЧИСЛО ПРОПУЩЕННЫХ САМОЛЕТОВ *
;*      (в левом верхнем углу экрана)     *
;*****************************************
PROC ShowEscapedPlanes NEAR
        pushad
        mov     SI,offset Font8x16
        mov     AX,[EscPlns]
        add     AX,'0'
        shl     AX,4
        add     SI,AX
        mov     EDI,4*LogicalStringLength+8
; Определить цвет цифры
```

продолжение ⇨

Листинг 4.18 (*продолжение*)

```
        mov     BL,LIGHTGREEN
        cmp     [EscPlns],0
        je      @@Bkgr
        mov     BL,YELLOW
        cmp     [EscPlns],1
        je      @@Bkgr
        mov     BL,LIGHTRED
@@Bkgr: mov     DL,BLUE
; Вывести цифру
        mov     AH,16 ;счетчик строк маски буквы
@@M0:   lodsb
        mov     CX,8  ;счетчик точек в строке маски
@@M1:   rol     AL,1
        jc      @@M2
        mov     [FS:EDI],DL
        mov     [FS:EDI+LogicalStringLength],DL
        inc     EDI
        mov     [FS:EDI],DL
        mov     [FS:EDI+LogicalStringLength],DL
        jmp     @@M3
@@M2:   mov     [FS:EDI],BL
        mov     [FS:EDI+LogicalStringLength],BL
        inc     EDI
        mov     [FS:EDI],BL
        mov     [FS:EDI+LogicalStringLength],BL
@@M3:   inc     EDI
        loop    @@M1
        add     EDI,2*LogicalStringLength-16
        dec     AH
        jnz     @@M0
        popad
        ret
ENDP ShowEscapedPlanes
ENDS
```

В листинге 4.19 собраны универсальные подпрограммы, которые могут использоваться как в одностраничном режиме, так и в режиме переключения страниц:

- процедура InitEpisode инициализирует переменные состояния динамических объектов в начале каждого эпизода — это просто линейный участок кода, вырезанный из основного модуля и оформленный в виде подпрограммы для большей наглядности;
- процедуры GShowDecByte, GShowDecWord и GShowDecDWord служат для вывода на экран в графическом режиме байта, слова и двойного слова данных в десятичном коде;

- подпрограмма ShowGameResults после завершения игры выводит на экран ее результаты, используя процедуры вывода десятичных чисел;
- процедуры CopyPlaneMask и MirrorPlaneMask предназначены для создания копии маски самолета, но CopyPlaneMask просто дублирует маску, а MirrorPlaneMask отражает ее слева направо (в начале каждого эпизода вызывается только одна из этих процедур — в зависимости от направления движения самолета; полученная таким образом копия маски используется в дальнейшем при выводе изображения самолета в видеопамять);
- процедура DrawMainBackground создает в дополнительной памяти компьютера фон изображения: верхняя область (небо) закрашивается в синий цвет, нижняя (земля) — в черный и зеленый (в клеточку); после этого поверх земли рисуются деревья в четыре ряда и пусковая установка в центре, а на небо наносятся облака (путем многократного наложения одной и той же маски);
- подпрограмма ExpMaskClear предназначена для очистки (обнуления) маски взрыва, а процедура ExplosionFrame — для генерации очередных кадров изображения взрыва (маска взрыва генерируется динамически путем наложения друг на друга цветных точек — осколков, разлетающихся в разные стороны с разными скоростями).

Листинг 4.19. Универсальные подпрограммы, пригодные для одностраничного и многостраничного режимов

```
CODESEG
;*************************************************
;*      ВЫВОД БАЙТА НА ЭКРАН В ДЕСЯТИЧНОМ КОДЕ    *
;* Подпрограмма выводит содержимое регистра AL    *
;* в десятичном коде в указанную позицию экрана.  *
;* Координаты позиции передаются через глобальные *
;* переменные ScreenString и ScreenColumn.        *
;*************************************************
PROC GShowDecByte NEAR
        push    EAX
        and     EAX,0FFh
        call    GShowDecDWord
        pop     EAX
        ret
ENDP GShowDecByte

;****************************************************
;*         ВЫВОД 16-РАЗРЯДНОГО СЛОВА НА ЭКРАН        *
;*                 В ДЕСЯТИЧНОМ КОДЕ                 *
```

продолжение ⇨

Листинг 4.19 (*продолжение*)

```
;* Параметры:                                        *
;* AX - число, которое будет выведено на экран.       *
;* Номер строки передается через глобальную           *
;* переменную ScreenString, номер столбца - через     *
;* переменную ScreenColumn, цвет текста определяется *
;* глобальной переменной TextColorAndBackground.       *
;*****************************************************
PROC GShowDecWord NEAR
        push    EAX
        and     EAX,0FFFFh
        call    GShowDecDWord
        pop     EAX
        ret
ENDP GShowDecWord

;*****************************************************
;*          ВЫВОД 32-РАЗРЯДНОГО СЛDВА НА ЭКРАН        *
;*                  В ДЕСЯТИЧНОМ КОДЕ                  *
;* Параметры:                                        *
;* EAX - число, которое будет выведено на экран.      *
;* Номер строки передается через глобальную           *
;* переменную ScreenString, номер столбца - через     *
;* переменную ScreenColumn, цвет текста определяется *
;* глобальной переменной TextColorAndBackground.       *
;*****************************************************
PROC GShowDecDWord NEAR
        pushad
        push    DS
; Настроить регистр DS на глобальный сегмент данных
        mov     SI,[CS:MainDataSeg]
        mov     DS,SI
; Перевести число в десятичный код
        mov     [Data_Int32],EAX
        call    Int32_to_String
        mov     DH,[byte ptr ScreenString]
        mov     DL,[byte ptr ScreenColumn]
; Вывести число на экран
        mov     CX,10
        mov     SI,offset Data_String
@@NextChar:
        lodsb               ;загрузить цифру в AL
        and     AL,AL       ;проверка на 0 (конец строки)
        jz      @@EndOfString
        call    PutGraChar
        loop    @@NextChar
@@EndDfString:
        pop     DS
        popad
        ret
```

```
ENDP GShowDecDWord

;****************************************************
;*  УСТАНОВИТЬ НАЧАЛЬНОЕ СОСТОЯНИЕ ПЕРЕМЕННЫХ ЭПИЗОДА  *
;* (вспомогательная процедура, регистры не сохраняет)  *
;****************************************************
PROC InitEpisode near
; Перерисовать фон
        call    ShowBackground
; Сбросить начальные адреса изображений объектов
        mov     [PlnA],0
        mov     [RktA],0
        mov     [FlmA],0
        mov     [ExpA],0
; Сбросить признаки наличия объектов
        mov     [PlnFl],0
        mov     [RktFl],0
        mov     [FlmFl],0
        mov     [ExpFl],0
; Сбросить счетчик времени
        mov     [GameTimeCounter],0
; Сбросить признак попадания ракеты в самолет
        mov     [HitFlag],0
; Установить начальное состояние самолета
        mov     [PlnState],0
        ; Прибавить случайное смещение по высоте
        mov     BX,[EpisodeNumber]
        shl     BX,2
        add     BX,offset DeltaS
        mov     EAX,[BX]
        add     EAX,80
        mov     [PlnS],EAX
        ; Запомнить направление движения самолета
        mov     BX,[EpisodeNumber]
        add     BX,offset Direction
        mov     AL,[BX]
        mov     [PlaneDirection],AL
        ; Вычислить скорость самолета
        mov     BX,[EpisodeNumber]
        shl     BX,2
        add     BX,offset PlnSpeed
        mov     EAX,[BX]
        mov     [PlaneSpeed],EAX
        ; Задержка пуска самолета
        mov     BX,[EpisodeNumber]
        shl     BX,2
        add     BX,offset DeltaT
        mov     AX,[BX]
        mov     [PlaneDeltaT],AX
        ; Скопировать маску самолета
```

продолжение ⇨

Листинг 4.19 (*продолжение*)

```
        call    CopyPlaneMask
        ; Сместить самолет за левую границу экрана
        mov     [PlnC],-PlnML
        ; Направление движения?
        cmp     [PlaneDirection],0
        je      @@dir0
        ; Сместить самолет за правую границу экрана
        mov     [PlnC],ScreenLength
        ; "Отрицательная" скорость
        neg     [PlaneSpeed]
        ; Использовать отраженную маску
        call    MirrorPlaneMask
; Установить начальное состояние ракеты
@@dir0: mov     [RktState],0 ;состояние 0
        ; Отобразить ракету на стартовой позиции
        mov     [RktFl],1  ;отобразить ракету
        mov     EAX,[PRS]  ;координаты стартовой позиции
        mov     [RktS],EAX
        mov     EAX,[PRC]
        mov     [RktC],EAX
        ret
ENDP InitEpisode

;*************************
;* ВЫВСТИ РЕЗУЛЬТАТЫ ИГРЫ *
;*************************
PROC ShowGameResults near
        pushad
; Очистить экрана
        call    GClearScreen
; Отобразить результаты
        mov     SI,offset EndTxt
        mov     [DefaultColor],LIGHTCYAN
        call    GShowString
        mov     [DefaultColor],LIGHTGREEN
        call    GShowString
        call    GShowString
        call    GShowString
        call    GShowString
        mov     [DefaultColor],YELLOW
        call    GShowString
        mov     [DefaultColor],WHITE
        mov     [ScreenString],12
        mov     [ScreenColumn],18
        mov     AX,[EpisodeNumber]
        call    GShowDecWord
        mov     [ScreenString],14
        mov     [ScreenColumn],16
        mov     AX,[RktCounter]
```

```
        call    GShowDecWord
        mov     [ScreenString],16
        mov     [ScreenColumn],17
        mov     AX,[DestroedPlns]
        call    GShowDecWord
        mov     [ScreenString],18
        mov     [ScreenColumn],21
        mov     AX,[EscPlns]
        call    GShowDecWord
; Ожидать нажатия любой клавиши
        call    GetChar
        popad
        ret
ENDP ShowGaшeResults

;************************
;* СОЗДАТЬ ОСНОВНОЙ ФОН *
;************************
PROC DrawMainBackground near
        pushad
; Загрузить начальное смещение (адрес видеостраницы)
        mov     EDI,VPage0Address
; Умножить высоту "неба" (высота экрана минус 64 строки)
; на логическую ширину строки (1024 пиксела)
        mov     ECX,ScreenHeigth-64
        shl     ECX,10
; Покрасить небо в синий цвет
        mov     AL,BLUE
@@NextPixel1:
        mov     [GS:EDI],AL
        inc     EDI
        dec     ECX
        jnz     @@NextPixel1
; Покрасить землю в черный и зеленый цвета
        mov     AL,BLACK
        mov     AH,GREEN
; Высота "земли" 64 строки
        mov     DX,64
@@NextStr:
        mov     CX,LogicalStringLength/2
@@NextPixel2:
        mov     [GS:EDI],AX
        add     EDI,2
        loop    @@NextPixel2
        xchg    AL,AH
        dec     DX
        jnz     @@NextStr

; НАРИСОВАТЬ ЛЕС
        ; Начальная строка "леса"
        mov     [dword ptr TreeS],ScreenHeigth-80
```

продолжение ⇗

Листинг 4.19 (*продолжение*)

```
        ; Число рядов деревьев
        mov     DX,4
; Цикл рисования леса
@@Trees:; Нарисовать очередной ряд деревьев
        mov     [dword ptr TreeC],0
        test    DX,01b
        jz      @@EvenLine
        ; Сместить нечетные ряды
        add     [dword ptr TreeC],16
@@EvenLine:
        mov     CX,ScreenLength/(TreeML*2)
@@NextTree:
        cmp     CX,ScreenLength/(TreeML*4)
        je      @@NoTree ;"просека" в лесу
        cmp     CX,ScreenLength/(TreeML*4)+1
        je      @@NoTree ;"просека" в лесу
        ; Нарисовать очередное дерево
DrawSImage TreeS,TreeC,TreeML,TreeMH,Tree
@@NoTree:
        add     [dword ptr TreeC],TreeML*2
        loop    @@NextTree
        add     [dword ptr TreeS],TreeMH/2
        dec     DX
        jnz     @@Trees

; НАРИСОВАТЬ ОБЛАКА
        mov     BX,offset CloudPos
        mov     CX,MaxCloudNum ;счетчик облаков
@@NextCloud:
        ; Нарисовать очередное облако
        mov     EAX,[BX]
        mov     [CloudS],EAX
        add     BX,4
        mov     EAX,[BX]
        mov     [CloudC],EAX
        add     BX,4
DrawSImage CloudS,CloudC,CloML,CloMH,Cloud
        loop    @@NextCloud

; НАРИСОВАТЬ ПУСКОВУЮ УСТАНОВКУ
        mov     [PRS],ScreenHeigth-PRMH-16
        mov     [PRC],(ScreenLength-PRML)/2
DrawSImage PRS,PRC,PRML,PRMH,Platf
        popad
        ret
ENDP DrawMainBackground

;*****************************
;* СКОПИРОВАТЬ МАСКУ САМОЛЕТА *
;*****************************
```

```
PROC CopyPlaneMask near
        pusha
        push    ES
        mov     AX,[CS:MainDataSeg]
        mov     ES,AX
        mov     SI,offset Plane
        mov     DI,offset PlnMask
        ; Скопировать весь массив
        mov     CX,PlnMH*PlnML
        rep     movsb
        pop     ES
        popa
        ret
ENDP CopyPlaneMask

;*************************************
;* ПОЛУЧИТЬ ОТРАЖЕНИЕ МАСКИ САМОЛЕТА *
;*************************************
PROC MirrorPlaneMask near
        pusha
        mov     SI,offset Plane
        mov     DI,offset PlnMask
        mov     DX,PlnMH
; Цикл по строкам
@@L0:   mov     CX,PlnML
        add     DI,PlnML-1
; Цикл по пикселам
@@L1:   lodsb
        mov     [DI],AL
        dec     DI
        loop    @@L1
        add     DI,PlnML+1
        dec     DX
        jnz     @@L0
        popa
        ret
ENDP MirrorPlaneMask

;************************
;* СТЕРЕТЬ МАСКУ ВЗРЫВА *
;************************
PROC ExpMaskClear near
        pusha
        push    ES
        mov     AX,[CS:MainDataSeg]
        mov     ES,AX
        mov     DI,offset ExpMask
        mov     CX,4*ExpR*ExpR
        mov     AL,0
        rep     stosb
```

продолжение ⇨

Листинг 4.19 (*продолжение*)

```
        pop     ES
        popa
        ret
ENDP ExpMaskClear

;****************************************
;* НАРИСОВАТЬ МАСКУ ОЧЕРЕДНОЙ ФАЗЫ ВЗРЫВА *
;****************************************
PROC ExplosionFrame near
        pusha
        cld
; Вывести изображение взрыва
        ; Записать адрес массива смещений осколков
        mov     SI,offset SplStep
        ; Записать число осколков
        mov     CX,SplinterMaxNumber
@@NextSplinter:
; Загрузить в DI указатель на маску взрыва
        mov     DI,offset ExpMask
; Умножить шаг по Y на номер кадра (плюс 1)
        lodsb           ;записать шаг по Y в AL
        mov     AH,[byte ptr ExpFrameNumber]
        inc     AH      ;увеличить номер кадра на 1
        imul    AH      ;умножить шаг на номер кадра
        ; Прибавить номер строки центра маски
        add     AX,ExpR-1
        ; Умножить разультат на длину строки маски
        mov     DX,2*ExpR
        mul     DX
        ; Прибавить результат к смещению в маске
        add     DI,AX
; Умножить шаг по X на номер кадра (плюс 1)
        lodsb           ;записать юаг по X в AL
        mov     AH,[byte ptr ExpFrameNumber]
        inc     AH      ;увеличить номер кадра на 1
        imul    AH      ;умножить юаг на номер кадра
        ; Прибавить номер колонки центра маски
        add     AX,ExpR-1
        ; Прибавить результат к смещению в маске
        add     DI,AX
; Нанести точку-осколок на маску
        mov     BX,offset ExpColors
        add     BX,[ExpFrameNumber]
        mov     AL,[BX]
        mov     [DI],AL
        loop    @@NextSplinter
        popa
        ret
ENDP ExplosionFrame
ENDS
```

Листинг 4.20 содержит головной модуль одностраничного варианта игры «Самолет и ракета» PlaneAndRocket. Перед запуском игры на экран выдается текст с описанием задания (сбить все пролетающие самолеты, при трех пропущенных игра заканчивается) и управляющих клавиш (Пробел — запуск ракеты, Esc — срочный выход из игры). Самолет движется по экрану с постоянной скоростью (1 или 2 пиксела на кадр); скорость, высота и направление полета выбираются случайным образом (на каждый параметр — своя таблица случайных чисел). Ракета имеет нелинейный начальный участок разгона, а затем движется с постоянной скоростью 3 пиксела на кадр. В каждом эпизоде игры имеется только одна ракета — поспешность или задержка с запуском приводят к проигрышу в данном эпизоде. В случае промаха ракета самоликвидируется в верхних слоях атмосферы (чтобы она не попадала в зону искажения изображений).

ПРИМЕЧАНИЕ

Под мертвую зону (область искажений) в верхней части экрана выделена полоса высотой в 32 пиксела (1/15 высоты экрана), куда не должен попасть ни один динамический объект. На медленных видеоконтроллерах такой ширины мертвой зоны недостаточно, и можно наблюдать разнообразные эффекты.

Листинг 4.20. Игра «Самолет и ракета» — основной модуль программы для одностраничного режима

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"
; Подключить описание констант, глобальных переменных
; и макрокоманд
include "list4_16.inc"
; Подключить файл масок объектов
include "list4_17.inc"

DATASEG
; ТЕКСТОВЫЕ СООБЩЕНИЯ
Txt1 DB 0,24,"АНИМАЦИЯ В ОДНОСТРАНИЧНОМ РЕЖИМЕ",0
     DB 2,2,"Параметры видеорежима: разрешение "
     DB "640x480, 256 цветов, линейная адресация",0
Txt2 DB 10,28,"ИГРА ",'"',"САМОЛЕТ И РАКЕТА",'"',0
```

продолжение

Листинг 4.20 (*продолжение*)

```
Txt3 DB 13,19
     DB "Задание: сбить все пролетающие самолеты.",0
     DB 15,22,"Игра завершается после 100 эпизодов",0
     DB 16,24,"или трех пропущенных самолетов.",0
Txt4 DB 20,0,"Управляющие клавиши:",0
Txt5 DB 22,0,"Пробел - запустить ракету;",0
     DB 24,0,"Esc    - срочный выход из программы.",0
Txt6 DB 29,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;*****************************************
;*            ОСНОВНАЯ ПРОГРАММА          *
;*****************************************
PROC PlaneAndRocket
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; "Захватить" текстовый шрифт
        call    GrabRusFont
; Установить видеорежим
        call    SetVESAVideoMode
; Установить режим прямой адресации памяти
        call    GInitialization

; ВВОДНЫЙ ТЕКСТ
; Отобразить текстовые сообщения
        mov     [DefaultBackground],BLACK ;черный фон
        mov     [DefaultColor],LIGHTGREEN ;зеленый текст
        MGShowText 2,Txt1
        mov     [DefaultColor],LIGHTCYAN  ;голубой текст
        MGShowString Txt2
        mov     [DefaultColor],LIGHTGREEN ;зеленый текст
        MGShowText 3,Txt3
        mov     [DefaultColor],YELLOW     ;желтый текст
        MGShowString Txt4
        mov     [DefaultColor],LIGHTGREEN ;зеленый текст
        MGShowText 2,Txt5
        mov     [DefaultColor],YELLOW     ;желтый текст
        MGShowString Txt6
        ; Ожидать нажатия любой клавиши
        call    GetChar
```

```
; ПОДГОТОВКА К ОСНОВНОМУ ЦИКЛУ
        call    DrawMainBackground ;рисование фона
; Сбросить счетчики
        mov     [PlnCounter],0
        mov     [DestroedPlns],0
        mov     [EscPlns],0
        mov     [RktCounter],0
        mov     [EpisodeNumber],0

;#####################
;# ЦИКЛ ПО ЭПИЗОДАМ #
;#####################
@@NextEpisode:
        call    InitEpisode

;#######################
;# ЦИКЛ ВЫВОДА КАДРА #
;#######################
@@FrameCycle:

; СТЕРЕТЬ ПОДВИЖНЫЕ ОБЪЕКТЫ
; Стереть самолет
DeleteMImage PlnA,PlnML,PlnMH,PlnF1
; Стереть ракету и пламя из ее сопла
DeleteMImage FlmA,FlmML,FlmMH,FlmF1
DeleteMImage RktA,RktML,RktMH,RktF1
; Стереть взрыв
DeleteMImage ExpA,ExpR*2,ExpR*2,ExpF1

; ОПРОС КЛАВИАТУРЫ
        call    WaitChar
        cmp     AX,0        ;клавиша была нажата?
        je      @@CommandNotInput ;нажатий не было
        cmp     AX,20h      ;нажат пробел?
        je      @@Fire      ;запустить ракету
        cmp     AX,1B00h    ;нажата клавиша ESC?
        je      @@Exit      ;выход
        jmp     @@CommandNotInput ;команды не было
; Пуск ракеты
@@Fire: cmp     [RktState],0
        jne     @@RocketNotReady ;ракета не готова
        mov     [RktStartFrameNumber],0
        mov     [RktState],1        ;запуск
        mov     [FlmF1],1           ;отобразить выхлоп
        inc     [RktCounter]
        mov     [GameStateGhange],1
@@RocketNotReady:
@@CommandNotInput:

; ВЫВОД ИЗОБРАЖЕНИЯ САМОЛЕТА
        cmp     [PlnState],0 ;ожидание
```

продолжение ⇒

Листинг 4.20 (*продолжение*)

```
        je      @@Pl0
        cmp     [PlnState],1 ;полет
        je      @@Pl1
        cmp     [PlnState],2 ;взрыв
        je      @@Pl2
; Ожидание самолета
@@Pl0:  ; Проверить истечение времени задержки самолета
        mov     AX,[PlaneDeltaT]
        cmp     [GameTimeCounter],AX
        jb      @@NoPl
        inc     [PlnCounter] ;увел. счетчик самолетов
        mov     [PlnState],1 ;запустить самолет
; Полет самолета
@@Pl1:  mov     [PlnF1],1    ;отобразить самолет
        mov     EAX,[PlaneSpeed]
        add     [PlnC],EAX
        jmp     @@NoPl
; Взрыв самолета
@@Pl2:  cmp     [ExpFrameNumber],MaxExplFrameNumber
        jae     @@EndOfEpisode
        ; Нарисовать маску очередной
        ; фазы взрыва самолета
        call    ExplosionFrame
        inc     [ExpFrameNumber]
@@NoPl: ; (завершена обработка состояния самолета)

; ВЫВОД ИЗОБРАЖЕНИЯ РАКЕТЫ
        cmp     [RktState],1 ;запуск ракеты
        je      @@ro1
        cmp     [RktState],2 ;ракета в полете
        je      @@ro2
        cmp     [RktState],3 ;взрыв ракеты
        je      @@ro3
        cmp     [RktState],4 ;разлет осколков
        je      @@ro4
        jmp     @@EndRktDraw
; Разгон ракеты
@@ro1:  mov     BX,[RktStartFrameNumber]
        cmp     BX,MaxRktStartFrame
        jb      @@NextRSFrame
        mov     [RktState],2
        jmp     @@ro2
@@NextRSFrame:
        shl     BX,3        ;умножить номер кадра на 8
        add     BX,offset RktStartStep
        mov     EAX,[PRC]
        mov     [RktC],EAX
        mov     EAX,[PRS]   ;вычесть смещение ракеты
        sub     EAX,[BX]    ;из номера строки установки
```

```
        mov     [RktS],EAX
        add     EAX,[BX+4] ;прибавить смещение пламени
        mov     [FlmS],EAX ;к номеру строки ракеты
        inc     [RktStartFrameNumber]
        jmp     @@EndRktDraw
; Полет ракеты
@@ro2:  ; Проверка на выход ракеты за верхнюю
        ; границу "атмосферы"
        cmp     [RktS],48   ;граница зоны искажений?
        jl      @@ro3       ;самоуничтожение ракеты
        ; Вычислить квадрат расстояния
        ; между самолетом и ракетой
        mov     EAX,[RktS]
        sub     EAX,[PlnS]
        imul    EAX
        ; (в EAX - квадрат расстояния по Y)
        mov     EBX,EAX
        mov     EAX,[RktC]
        sub     EAX,[PlnC]
        sub     EAX,(PlnML-RktML)/2
        imul    EAX         ;квадрат расстояния по X
        ; (в EAX - квадрат расстояния по X)
        add     EAX,EBX
        ; (в EAX - квадрат расстояния между самолетом
        ; и ракетой)
        cmp     EAX,TargetDistanceSQ
        jae     @@DrawRocket
        ; Произошло попадание ракеты в самолет
        mov     [HitFlag],1 ;установить флаг попадания
        jmp     @@ro3       ;начать отображение взрыва
@@DrawRocket:
        sub     [RktS],3
        mov     EAX,[RktS]
        add     EAX,32
        mov     [FlmS],EAX
        jmp     @@EndRktDraw

; Начало взрыва ракеты
@@ro3:  ; Переключить состояние ракеты
        mov     [RktState],4
        ; Очистить маску взрыва
        call    ExpMaskClear
        ; Определить координаты взрыва
        mov     [ExpFrameNumber],0
        mov     EAX,[RktS]
        sub     EAX,ExpR-RktMH/2
        mov     [ExpS],EAX
        mov     EAX,[RktC]
        sub     EAX,ExpR-RktML/2
        mov     [ExpC],EAX
```

продолжение ⇒

Листинг 4.20 (*продолжение*)

```
        mov     [RktFl],0     ;убрать ракету
        mov     [FlmFl],0     ;убрать выхлоп
        mov     [ExpFl],1     ;отобразить взрыв

; Разлет осколков ракеты
@@ro4:  cmp     [ExpFrameNumber],MaxExplFrameNumber
        jae     @@SetRktState5
        ; Нарисовать маску очередной фазы взрыва ракеты
        call    ExplosionFrame
        inc     [ExpFrameNumber]
        jmp     @@EndRktDraw
@@SetRktState5:
        mov     [RktState],5 ;перезарядка установки
        mov     [ExpFl],0     ;убрать взрыв
        ; Было попадание в самолет?
        cmp     [HitFlag],0
        je      @@EndRktDraw ;попадания не было
        mov     [PlnState],2 ;взрыв самолета
        ; Увеличить значение счетчика сбитых самолетов
        inc     [DestroedPlns]
        mov     [GameStateGhange],1
        ; Очистить маску взрыва
        call    ExpMaskClear
        ; Определить координаты взрыва
        mov     [ExpFrameNumber],0
        mov     EAX,[PlnS]
        sub     EAX,ExpR-PlnMH/2
        mov     [ExpS],EAX
        mov     EAX,[PlnC]
        sub     EAX,ExpR-PlnML/2
        mov     [ExpC],EAX
        mov     [ExpFl],1     ;отобразить взрыв самолета
@@EndRktDraw:

; ОТОБРАЗИТЬ ДИНАМИЧЕСКИЕ ОБЪЕКТЫ
; Нарисовать самолет
DrawMImage PlnS,PlnC,PlnML,PlnMH,PlnMask,PlnA,PlnFl
; Нарисовать пламя ракеты
DrawMImage FlmS,RktC,FlmML,FlmMH,Flm,FlmA,FlmFl
; Нарисовать ракету
DrawMImage RktS,RktC,RktML,RktMH,Rkt,RktA,RktFl
; Отобразить взрыв
DrawMImage ExpS,ExpC,ExpR*2,ExpR*2,ExpMask,ExpA,ExpFl
; Счетчик пропущенных самолетов
        call    ShowEscapedPlanes

; ОЖИДАНИЕ НАЧАЛА СЛЕДУЮЩЕГО КАДРА
        call    WaitVSync
        ; Увеличить на 1 счетчик времени
```

```
        inc     [GameTimeCounter]

; КОНЕЦ ЦИКЛА ПО КАДРАМ
        ; Самолет покинул пределы экрана?
        cmp     [PlaneDirection],0
        je      @@LtoR          ;движение слева направо
        cmp     [PlnC],-PlnML
        jl      @@PlaneEscaped
        jmp     @@FrameCycle
@@LtoR: cmp     [PlnC],ScreenLength
        jl      @@FrameCycle
@@PlaneEscaped:
; Увеличить счетчик пропущенных самолетов
        inc     [EscPlns]
        mov     [GameStateGhange],1

; КОНЕЦ ЦИКЛА ПО ЭПИЗОДАМ
@@EndOfEpisode:
; Увеличить счетчик игровых эпизодов
        inc     [EpisodeNumber]
; Закончить игру после MaxEpNumber эпизодов
        cmp     [EpisodeNumber],MaxEpNumber
        jae     @@Exit
; Конец игры, если пропущено MaxSavedPlanes самолетов
        cmp     [EscPlns],MaxSavedPlanes
        jl      @@NextEpisode

@@Exit:
; Показать результаты
        call    ShowGameResults
; Установить текстовый режим
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP PlaneAndRocket
ENDS

; Подключить процедуры вывода данных на экран
; для текстовых режимов
include "list1_02.inc"
; Подключить процедуры перевода чисел
include "list2_05.inc"
; Подключить набор процедур общего назначения,
; предназначенных для установки графических
; видеорежимов и работы в них
include "list4_02.inc"
; Подключить подпрограмму, настраивающую FS на
; видеопамять, GS - на дополнительную память
include "list4_14.inc"
```

продолжение

Листинг 4.20 (*продолжение*)

```
; Подключить набор процедур вывода текста,
; предназначенный для 256-цветных режимов с
; раздельным доступом с сегментам видеопамяти
; и дополнительной памяти
include "list4_15.inc"
; Подключить набор процедур для вывода
; изображений объектов в одностраничном режиме
include "list4_18.inc"
; Подключить набор универсальных подпрограмм, пригодных
; и для одностраничного, и для двухстраничного режима
include "list4_19.inc"

END
```

Листинг 4.21 содержит модернизированный для режима переключения страниц вариант листинга 4.18: внесены изменения в процедуры DrawMovingImage, DeleteImage, ShowBackground, ShowEscapedPlanes; добавлены процедура переключения видеостраниц SwitchVideoPage, процедура возврата в одностраничный режим работы RestoreNormalMode и процедура инициализации дополнительных переменных эпизода InitEpisode2. Все изменения в процедурах связаны с тем, что адрес страницы видеопамяти из константы превращается в переменную.

ПРИМЕЧАНИЕ

Обратите внимание на то, что структура области сохранения фона в дополнительной памяти в точности повторяет собой структуру видеостраниц. Если у видеостраницы есть невидимые защитные полосы, то такие же полосы есть и у массива сохранения фона. Подобная согласованность структур часто бывает необходима, чтобы избежать лишних перерасчетов и перекодировок.

Листинг 4.21. Набор подпрограмм для вывода фона и динамических объектов в режиме переключения страниц

```
; Номера начальных строк видеостраниц
VPage0StartString equ 64
VPage1StartString equ 608
; Начальные адреса видеостраниц
VPage0Address     equ 65536
VPage1Address     equ 622592

; Начальные адрес строки статуса в оперативной памяти
; (располагается вслед за областью сохранения фона)
StatusStringAddress equ B0000h

DATASEG
```

```
; ОБЩИЕ ПАРАМЕТРЫ ВИДЕОРЕЖИМА
; Номер перерисовываемой видеостраницы
VPageNumber DB 0
; Адрес перерисовываемой видеостраницы
DrPageAddress DD VPage0Address
; Адрес отображаемой видеостраницы
ShPageAddress DD VPage0Address
; ПАРАМЕТРЫ ПОДПРОГРАММЫ РИСОВАНИЯ СПРАЙТА
; Указатели на маску и область сохранения фона
ImageMaskOffset DW ? ;указатель на маску
ImageBackOffset DW ? ;указатель на фон
; Размеры маски изображения
ImageL DD ? ;ширина маски
ImageH DD ? ;высота маски
; Позиция маски изображения на экране
ImageS DD ? ;строка
ImageC DD ? ;колонка
ImageA DD ? ;линейный адрес
ImageF DB ? ;флаг присутствия объекта на странице
ENDS

CODESEG
;*********************************************
;*          ПЕРЕКЛЮЧИТЬ ВИДЕОСТРАНИЦЫ        *
;* Процедура использует в качестве параметров *
;* глобальные переменные VPageNumber,        *
;* DrPageAddress, ShPageAddress.             *
;*********************************************
PROC SwitchVideoPage NEAR
        pusha
        cmp     [VPageNumber],0
        jne     @@Pg1
        mov     [VPageNumber],1
        mov     [DrPageAddress],VPage1Address
        mov     [ShPageAddress],VPage0Address
        mov     DX,VPage0StartString
        jmp short @@UseVESAFunction
@@Pg1:  mov     [VPageNumber],0
        mov     [DrPageAddress],VPage0Address
        mov     [ShPageAddress],VPage1Address
        mov     DX,VPage1StartString
@@UseVESAFunction:
        mov     AX,4F07h
        mov     BX,0
        mov     CX,0
        int     10h
        popa
        ret
ENDP SwitchVideoPage
```

продолжение ⇒

Листинг 4.21 (*продолжение*)

```
;***********************************
;* ВОССТАНОВИТЬ ОДНОСТРАНИЧНЫЙ РЕЖИМ *
;*  (процедура параметров не имеет)  *
;***********************************
PROC RestoreNormalMode NEAR
        pusha
        mov     AX,4F07h
        mov     BX,0
        mov     CX,0
        mov     DX,0
        int     10h
        popa
        ret
ENDP RestoreNormalMode

;*************************************
;* ОТОБРАЗИТЬ ФОН НА ОБЕ ВИДЕОСТРАНИЦЫ *
;*   (процедура параметров не имеет)   *
;*************************************
PROC ShowBackground near
        pushad
        cld
        ; Настроить ESI на область сохранения фона
        mov     ESI,VPage0Address
        ; Настроить EDI на первую видеостраницу
        mov     EDI,VPage0Address
        ; Загрузить в счетчик размер изображения
        mov     ECX,LogicalStringLength*ScreenHeigth
        ; Скопировать фон в обе страницы сразу
@@N:    mov     AL,[GS:ESI]
        mov     [FS:EDI],AL
        mov     [FS:EDI+VPage1Address-VPage0Address],AL
        inc     ESI
        inc     EDI
        dec     ECX
        jnz     @@N
        popad
        ret
ENDP ShowBackground

;************************************************
;*          НАРИСОВАТЬ ДИНАМИЧЕСКИЙ ОБЪЕКТ         *
;* Для передачи параметров используются глобальные *
;* переменные ImageMaskOffset, ImageL, ImageH,    *
;* ImageS, ImageC, ImageA, ImageF, DrPageAddress.  *
;************************************************
PROC DrawMovingImage near
        pushad
        cmp     [ImageF],0
```

```
        je      @@End
        cld
; Вычислить адрес начальной точки для вывода маски
        ; Умножить длину строки на номер строки (Y)
        mov     EAX,LogicalStringLength
        mov     EDX,[ImageS]
        mul     EDX
        ; Прибавить номер колонки (X)
        add     EAX,[ImageC]
        mov     [ImageA],EAX ;запомнить смещение
        ; Прибавить смещение перерисовываемой страницы
        add     EAX,[DrPageAddress]
        mov     EDI,EAX ;результат - в индексный регистр
        ; Записать адрес маски в индексный регистр
        mov     SI,[ImageMaskOffset]
; Вывести изображение
        mov     DX,[word ptr ImageH] ;высота маски
@@M0:   ; Вывести очередную строку маски
        mov     CX,[word ptr ImageL] ;ширина маски
@@M1:   ; Проверить точку маски
        lodsb
        and     AL,AL         ;код цвета равен нулю?
        jz      @@M2          ;пропустить точку
        mov     [FS:EDI],AL  ;вывести точку
@@M2:   ; Перейти к следующей точке
        inc     EDI
        loop    @@M1
        ; Перейти на следующую строку
        add     EDI,LogicalStringLength
        sub     EDI,[ImageL]
        dec     DX
        jnz     @@M0
@@End:  popad
        ret
ENDP DrawMovingImage

;****************************************************
;*              НАРИСОВАТЬ ЭЛЕМЕНТ ФОНА             *
;* Для передачи параметров используются глобальные *
;* переменные ImageMaskOffset, ImageL, ImageH,      *
;* ImageS, ImageC.                                  *
;****************************************************
PROC DrawStaticImage near
        pushad
        cld
; Загрузить начальное смещение
        mov     EDI,VPage0Address
; Вычислить адрес начальной точки для вывода маски
        ; Умножить длину строки на номер строки (Y)
        mov     EAX,LogicalStringLength
        mov     EDX,[ImageS]
```

продолжение ⇨

Листинг 4.21 (*продолжение*)

```
        mul     EDX
        ; Прибавить номер колонки (X)
        add     EAX,[ImageC]
        ; Прибавить результат к начальному смещению
        add     EDI,EAX
        ; Записать адрес маски в индексный регистр
        mov     SI,[ImageMaskDffset]
; Вывести изображение
        mov     DX,[word ptr ImageH] ;высота маски
@@M0:   ; Вывести очередную строку маски
        mov     CX,[word ptr ImageL] ;ширина маски
@@M1:   ; Проверить точку маски
        lodsb
        and     AL,AL          ;код цвета равен нулю?
        jz      @@M2           ;пропустить точку
        mov     [GS:EDI],AL    ;вывести точку
@@M2:   ; Перейти к следующей точке
        inc     EDI
        loop    @@M1
        ; Перейти на следующую строку
        add     EDI,LogicalStringLength
        sub     EDI,[ImageL]
        dec     DX
        jnz     @@M0
        popad
        ret
ENDP DrawStaticImage

;***************************************************
;*      СТЕРЕТЬ ИЗОБРАЖЕНИЕ (ВОССТАНОВИТЬ ФОН)      *
;* Для передачи параметров используются глобальные *
;* переменные ImageL, ImageH, ImageA, ImageF,      *
;* DrPageAddress.                                  *
;***************************************************
PROC DeleteImage near
        pushad
        cmp     [ImageF],0
        je      @@End
; Записать в индексные регистры адрес изображения
        mov     EDI,[ImageA]
        mov     ESI,EDI
; Прибавить начальное смещение
        add     EDI,[DrPageAddress]
        add     ESI,VPage0Address
; Вывести исходное изображение
        mov     DX,[word ptr ImageH] ;высота маски
@@M0:   mov     CX,[word ptr ImageL] ;ширина маски
@@M1:   mov     AL,[GS:ESI]
        mov     [FS:EDI],AL
```

```
        inc     EDI
        inc     ESI
        loop    @@M1
        add     EDI,LogicalStringLength
        sub     EDI,[ImageL]
        add     ESI,LogicalStringLength
        sub     ESI,[ImageL]
        dec     DX
        jnz     @@M0
@@End:  popad
        ret
ENDP DeleteImage

;*****************************************
;* ОТОБРАЗИТЬ ЧИСЛО ПРОПУЩЕННЫХ САМОЛЕТОВ *
;*      (в левом верхнем углу экрана)    *
;*****************************************
PROC ShowEscapedPlanes NEAR
        pushad
        mov     SI,offset FontBx16
        mov     AX,[EscPlns]
        add     AX,'0'
        shl     AX,4
        add     SI,AX
        mov     EDI,[DrPageAddress]
        add     EDI,4*LogicalStringLength+8
; Определить цвет цифры
        mov     BL,LIGHTGREEN
        cmp     [EscPlns],0
        je      @@Bkgr
        mov     BL,YELLOW
        cmp     [EscPlns],1
        je      @@Bkgr
        mov     BL,LIGHTRED
@@Bkgr: mov     DL,BLUE
; Вывести цифру
        mov     AH,16 ;счетчик строк маски буквы
@@M0:   lodsb
        mov     CX,8  ;счетчик точек в строке маски
@@M1:   rol     AL,1
        jc      @@M2
        mov     [FS:EDI],DL
        mov     [FS:EDI+LogicalStringLength],DL
        inc     EDI
        mov     [FS:EDI],DL
        mov     [FS:EDI+LogicalStringLength],DL
        jmp     @@M3
@@M2:   mov     [FS:EDI],BL
        mov     [FS:EDI+LogicalStringLength],BL
        inc     EDI
        mov     [FS:EDI],BL
```

продолжение ⇗

Листинг 4.21 (*продолжение*)

```
        mov     [FS:EDI+LogicalStringLength],BL
@@M3:   inc     EDI
        loop    @@M1
        add     EDI,2*LogicalStringLength-16
        dec     AH
        jnz     @@M0
        popad
        ret
ENDP ShowEscapedPlanes

;****************************************************
;* УСТАНОВИТЬ НАЧАЛЬНОЕ СОСТОЯНИЕ ПЕРЕМЕННЫХ ЭПИЗОДА *
;****************************************************
PROC InitEpisode2 near
        call    InitEpisode
        mov     [PlnF2],0
        mov     [RktF2],0
        mov     [FlmF2],0
        mov     [ExpF2],0
        call    SwitchVideoPage
        ret
ENDP InitEpisode2
ENDS
```

Листинг 4.22 содержит головной модуль игры «Самолет и ракета» для режима с переключением видеостраниц PlaneAndRocket2. Нормальная работа этого варианта игры возможна только при наличии у видеоконтроллера не менее 2 Мбайт памяти.

Листинг 4.22. Игра «Самолет и ракета» — основной модуль программы для режима с переключением страниц

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"
; Подключить описание констант, глобальных переменных
; и макрокоманд
include "list4_16.inc"
; Подключить файл масок объектов
include "list4_17.inc"

DATASEG
```

```
; ИНФОРМАЦИЯ ОБ ОТОБРАЖАЕМЫХ ОБЪЕКТАХ
; Флаги наличия объектов в предыдущей странице
PlnF2  DB ?
RktF2  DB ?
FlmF2  DB ?
ExpF2  DB ?
; Предыдущие линейные адреса объектов
PlnAP  DD ?
RktAP  DD ?
FlmAP  DD ?
ExpAP  DD ?
; ТЕКСТОВЫЕ СООБЩЕНИЯ
Txt1 DB 0,21,"АНИМАЦИЯ В РЕЖИМЕ ПЕРЕКЛЮЧЕНИЯ СТРАНИЦ",0
     DB 2,2,"Параметры видеорежима: разрешение "
     DB "640x480, 256 цветов, линейная адресация",0
Txt2 DB 10,28,"ИГРА ",'"',"САМОЛЕТ И РАКЕТА",'"',0
Txt3 DB 13,19
     DB "Задание: сбить все пролетающие самолеты.",0
     DB 15,22,"Игра завершается после 100 эпизодов",0
     DB 16,24,"или трех пропущенных самолетов.",0
Txt4 DB 20,0,"Управляющие клавиши:",0
Txt5 DB 22,0,"Пробел - запустить ракету;",0
     DB 24,0,"Esc    - срочный выход из программы.",0
Txt6 DB 29,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;*****************************************
;*           ОСНОВНАЯ ПРОГРАММА          *
;*****************************************
PROC PlaneAndRocket2
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; "Захватить" текстовый шрифт
        call    GrabRusFont
; Установить видеорежим
        call    SetVESAVideoMode
; Установить режим прямой адресации памяти
        call    GInitialization

; ВВОДНЫЙ ТЕКСТ
; Отобразить текстовые сообщения
        mov     [DefaultBackground],BLACK ;черный фон
```

продолжение

Листинг 4.22 (*продолжение*)

```
        mov     [DefaultColor],WHITE      ;белый текст
        MGShowText 2,Txt1
        mov     [DefaultColor],LIGHTCYAN  ;голубой текст
        MGShowString Txt2
        mov     [DefaultColor],LIGHTGREEN ;зеленый текст
        MGShowText 3,Txt3
        mov     [DefaultColor],YELLOW     ;желтый текст
        MGShowString Txt4
        mov     [DefaultColor],LIGHTGREEN ;зеленый текст
        MGShowText 2,Txt5
        mov     [DefaultColor],YELLOW     ;желтый текст
        MGShowString Txt6
        ; Ожидать нажатия любой клавиши
        call    GetChar

; ПОДГОТОВКА К ОСНОВНОМУ ЦИКЛУ
        call    DrawMainBackground ;создать фон
; Сбросить счетчики
        mov     [PlnCounter],0
        mov     [DestroedPlns],0
        mov     [EscPlns],0
        mov     [RktCounter],0
        mov     [EpisodeNumber],0

;###################
;# ЦИКЛ ПО ЭПИЗОДАМ #
;###################
@@NextEpisode:
        call    InitEpisode2

;####################
;# ЦИКЛ ВЫВОДА КАДРА #
;####################
@@FrameCycle:

; СТЕРЕТЬ ПОДВИЖНЫЕ ОБЪЕКТЫ
; Стереть самолет
DeleteMImage PlnAP,PlnML,PlnMH,PlnF2
; Стереть ракету и пламя из ее сопла
DeleteMImage FlmAP,FlmML,FlmMH,FlmF2
DeleteMImage RktAP,RktML,RktMH,RktF2
; Стереть взрыв
DeleteMImage ExpAP,ExpR*2,ExpR*2,ExpF2

; ПЕРЕРИСОВЫВАТЬ ПОДВИЖНЫЕ ОБЪЕКТЫ
; Скопировать начальные адреса в "предыдущие"
        mov     EAX,[PlnA]
        mov     [PlnAP],EAX
        mov     EAX,[RktA]
```

```
        mov     [RktAP],EAX
        mov     EAX,[FlmA]
        mov     [FlmAP],EAX
        mov     EAX,[ExpA]
        mov     [ExpAP],EAX
; Скопировать признаки наличия объектов в "предыдущие"
        mov     AL,[PlnF1]
        mov     [PlnF2],AL
        mov     AL,[RktF1]
        mov     [RktF2],AL
        mov     AL,[FlmF1]
        mov     [FlmF2],AL
        mov     AL,[ExpF1]
        mov     [ExpF2],AL
; ОПРОС КЛАВИАТУРЫ
        call    WaitChar
        cmp     AX,0        ;клавиша была нажата?
        je      @@CommandNotInput ;нажатий не было
        cmp     AX,20h      ;нажат пробел?
        je      @@Fire      ;запустить ракету
        cmp     AX,1B00h    ;нажата клавиша ESC?
        je      @@Exit      ;выход
        jmp     @@CommandNotInput ;команды не было
; Пуск ракеты
@@Fire: cmp     [RktState],0
        jne     @@RocketNotReady ;ракета не готова
        mov     [RktStartFrameNumber],0
        mov     [RktState],1        ;запуск
        mov     [FlmF1],1           ;отобразить выхлоп
        inc     [RktCounter]
        mov     [GameStateGhange],1
@@RocketNotReady:
@@CommandNotInput:

; ВЫВОД ИЗОБРАЖЕНИЯ САМОЛЕТА
        cmp     [PlnState],0 ;ожидание
        je      @@Pl0
        cmp     [PlnState],1 ;полет
        je      @@Pl1
        cmp     [PlnState],2 ;взрыв
        je      @@Pl2
; Ожидание самолета
@@Pl0:  ; Проверить истечение времени задержки самолета
        mov     AX,[PlaneDeltaT]
        cmp     [GameTimeCounter],AX
        jb      @@NoPl
        inc     [PlnCounter] ;увеличить счетчик самолетов
        mov     [PlnState],1 ;запустить самолет
; Полет самолета
@@Pl1:  mov     [PlnF1],1    ;отобразить самолет
        mov     EAX,[PlaneSpeed]
```

продолжение ⇨

Листинг 4.22 (*продолжение*)

```
        add     [PlnC],EAX
        jmp     @@NoPl
; Взрыв самолета
@@Pl2:  cmp     [ExpFrameNumber],MaxExplFrameNumber
        jae     @@EndOfEpisode
        ; Нарисовать маску очередной
        ; фазы взрыва самолета
        call    ExplosionFrame
        inc     [ExpFrameNumber]
@@NoPl: ; (завершена обработка состояния самолета)

; ВЫВОД ИЗОБРАЖЕНИЯ РАКЕТЫ
        cmp     [RktState],1 ;запуск ракеты
        je      @@ro1
        cmp     [RktState],2 ;ракета в полете
        je      @@ro2
        cmp     [RktState],3 ;взрыв ракеты
        je      @@ro3
        cmp     [RktState],4 ;разлет осколков
        je      @@ro4
        jmp     @@EndRktDraw
; Разгон ракеты
@@ro1:  mov     BX,[RktStartFrameNumber]
        cmp     BX,MaxRktStartFrame
        jb      @@NextRSFrame
        mov     [RktState],2
        jmp     @@ro2
@@NextRSFrame:
        shl     BX,3        ;умножить номер кадра на 8
        add     BX,offset RktStartStep
        mov     EAX,[PRC]
        mov     [RktC],EAX
        mov     EAX,[PRS]  ;вычесть смещение ракеты
        sub     EAX,[BX]   ;из номера строки установки
        mov     [RktS],EAX
        add     EAX,[BX+4] ;прибавить смещение пламени
        mov     [FlmS],EAX ;к номеру строки ракеты
        inc     [RktStartFrameNumber]
        jmp     @@EndRktDraw
; Полет ракеты
@@ro2:  ; Проверка на выход ракеты за верхнюю
        ; границу "атмосферы"
        cmp     [RktS],0
        jl      @@ro3       ;самоуничтожение ракеты
        ; Вычислить квадрат расстояния между
        ; самолетом и ракетой
        mov     EAX,[RktS]
        sub     EAX,[PlnS]
        imul    EAX
```

```
        ; (в EAX - квадрат расстояния по Y)
        mov     EBX,EAX
        mov     EAX,[RktC]
        sub     EAX,[PlnC]
        sub     EAX,(PlnML-RktML)/2
        imul    EAX           ;квадрат расстояния по X
        ; (в EAX - квадрат расстояния по X)
        add     EAX,EBX
        ; (в EAX - квадрат расстояния между самолетом
        ; и ракетой)
        cmp     EAX,TargetDistanceSQ
        jae     @@DrawRocket
        ; Произошло попадание ракеты в самолет
        mov     [HitFlag],1 ;установить флаг попадания
        jmp     @@ro3       ;начать отображение взрыва
@@DrawRocket:
        sub     [RktS],3
        mov     EAX,[RktS]
        add     EAX,32
        mov     [FlmS],EAX
        jmp     @@EndRktDraw

; Начало взрыва ракеты
@@ro3:  ; Переключить состояние ракеты
        mov     [RktState],4
        ; Очистить маску взрыва
        call    ExpMaskClear
        ; Определить координаты взрыва
        mov     [ExpFrameNumber],0
        mov     EAX,[RktS]
        sub     EAX,ExpR-RktMH/2
        mov     [ExpS],EAX
        mov     EAX,[RktC]
        sub     EAX,ExpR-RktML/2
        mov     [ExpC],EAX
        mov     [RktF1],0    ;убрать ракету
        mov     [FlmF1],0    ;убрать выхлоп
        mov     [ExpF1],1    ;отобразить взрыв

; Разлет осколков ракеты
@@ro4:  cmp     [ExpFrameNumber],MaxExplFrameNumber
        jae     @@SetRktState5
        ; Нарисовать маску очередной фазы взрыва ракеты
        call    ExplosionFrame
        inc     [ExpFrameNumber]
        jmp     @@EndRktDraw
@@SetRktState5:
        mov     [RktState],5 ;перезарядка установки
        mov     [ExpF1],0    ;убрать взрыв
        ; Было попадание в самолет?
        cmp     [HitFlag],0
```

продолжение ⇒

Листинг 4.22 (*продолжение*)

```
        je      @@EndRktDraw ;попадания не было
        mov     [PlnState],2 ;взрыв самолета
        ; Увеличить значение счетчика сбитых самолетов
        inc     [DestroedPlns]
        mov     [GameStateGhange],1
        ; Очистить маску взрыва
        call    ExpMaskClear
        ; Определить координаты взрыва
        mov     [ExpFrameNumber],0
        mov     EAX,[PlnS]
        sub     EAX,ExpR-PlnMH/2
        mov     [ExpS],EAX
        mov     EAX,[PlnC]
        sub     EAX,ExpR-PlnML/2
        mov     [ExpC],EAX
        mov     [ExpF1],1    ;отобразить взрыв самолета
@@EndRktDraw:

; ОТОБРАЗИТЬ ДИНАМИЧЕСКИЕ ОБЪЕКТЫ
; Нарисовать самолет
DrawMImage PlnS,PlnC,PlnML,PlnMH,PlnMask,PlnA,PlnF1
; Нарисовать пламя ракеты
DrawMImage FlmS,RktC,FlmML,FlmMH,Flm,FlmA,FlmF1
; Нарисовать ракету
DrawMImage RktS,RktC,RktML,RktMH,Rkt,RktA,RktF1
; Отобразить взрыв
DrawMImage ExpS,ExpC,ExpR*2,ExpR*2,ExpMask,ExpA,ExpF1
; Счетчик пропущенных самолетов
        call    ShowEscapedPlanes

; ОЖИДАНИЕ НАЧАЛА СЛЕДУЮЩЕГО КАДРА
        call    WaitVSync
        ; Переключить страницы
        call    SwitchVideoPage
        ; Увеличить на 1 счетчик времени
        inc     [GameTimeCounter]

; КОНЕЦ ЦИКЛА ПО КАДРАМ
        ; Самолет покинул пределы экрана?
        cmp     [PlaneDirection],0 ;направление движения?
        je      @@LtoR
        cmp     [PlnC],-PlnML
        jl      @@PlaneEscaped
        jmp     @@FrameCycle
@@LtoR: cmp     [PlnC],ScreenLength
        jl      @@FrameCycle
@@PlaneEscaped:
; Увеличить счетчик пропущенных самолетов
        inc     [EscPlns]
```

```
        mov     [GameStateGhange],1

; КОНЕЦ ЦИКЛА ПО ЭПИЗОДАМ
@@EndOfEpisode:
; Увеличить счетчик игровых эпизодов
        inc     [EpisodeNumber]
; Закончить игру после MaxEpNumber эпизодов
        cmp     [EpisodeNumber],MaxEpNumber
        jae     @@Exit
; Конец игры, если пропущено MaxSavedPlanes самолетов
        cmp     [EscPlns],MaxSavedPlanes
        jl      @@NextEpisode

@@Exit:
; Восстановить обычный режим работы с видеопамятью
        call    RestoreNormalMode
; Показать результаты
        call    ShowGameResults
; Установить текстовый режим
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP PlaneAndRocket2
ENDS

; Подключить процедуры вывода данных на экран
; для текстовых режимов
include "list1_02.inc"
; Подключить процедуры перевода чисел
include "list2_05.inc"
; Подключить набор процедур общего назначения,
; предназначенных для установки графических
; видеорежимов и работы в них
include "list4_02.inc"
; Подключить подпрограмму, настраивающую FS на
; видеопамять, GS - на дополнительную память
include "list4_14.inc"
; Подключить набор процедур вывода текста,
; предназначенный для 256-цветных режимов с
; раздельным доступом с сегментам видеопамяти
; и дополнительной памяти
include "list4_15.inc"
; Подключить набор процедур для вывода изображений
; объектов в режиме переключения страниц
include "list4_21.inc"
; Подключить набор универсальных подпрограмм, пригодных
; и для одностраничного, и для двухстраничного режима
include "list4_19.inc"

END
```

Простые форматы графических файлов

Задача вывода изображений на экран тесно связана с двумя другими задачами — сохранения изображений на диске в виде файлов и считывания изображений с диска. Существует множество различных форматов графических файлов [9, 25, 28], однако эти форматы отличаются обычно значительной сложностью. Далее мы рассмотрим два самых простых формата файлов — формат BMP (для несжатого RGB-изображения) и формат PCX для 256-цветных режимов.

Формат BMP для несжатого RGB-изображения

Формат файла BMP для несжатого RGB-изображения показан в табл. 4.10. Это самый простой из всех форматов, он воспринимается большинством графических редакторов, но порождает файлы гигантских размеров, поскольку одна точка изображения кодируется тремя байтами данных, по байту на каждый из цветовых компонентов (аналогично формату TrueColor24).

Таблица 4.10. Формат файла BMP для несжатого RGB-изображения

Смещение	Размер	Значение	Описание
00h	WORD	4D42h (BM)	Признак файла BMP
02h	DWORD	3xLxH + 54	Полный размер файла в байтах
06h	WORD	0	Не используется
08h	WORD	0	Не используется
0Ah	DWORD	54 (36h)	Смещение области данных изображения от начала файла
0Eh	DWORD	40 (28h)	Размер описателя изображения
12h	DWORD	L	Ширина изображения в пикселах
16h	DWORD	H	Высота изображения в пикселах
1Ah	WORD	1	Число битовых плоскостей
1Ch	WORD	24 (18h)	Число битов на пиксел
1Eh	DWORD	0	Метод сжатия
22h	DWORD	3xLxH	Размер изображения в байтах
26h	DWORD	0	Разрешение по горизонтали в пикселах на метр

Смещение	Размер	Значение	Описание
2Ah	DWORD	0	Разрешение по вертикали в пикселах на метр
2Eh	DWORD	0	Число цветов в растровом изображении
32h	DWORD	0	Число важных цветов изображения
36h	3xLxH байт	—	Область данных изображения

Создание файла BMP начинается с заполнения заголовка. Как видно из таблицы, в формате RGB большая часть полей заголовка заполняется строго определенными значениями (константами), никак не зависящими от вида и размеров изображения. Переменными параметрами являются только ширина изображения L, высота изображения H, размер изображения в байтах (равен умноженному на три произведению ширины и высоты) и полный размер файла в байтах (является суммой размера изображения и размера заголовка).

Допустим, нужно сохранить изображение размером 1024×768 точек. Тогда ширина L=1024, высота H=768, размер изображения равен 2 359 296 байт, а размер файла — 2 359 350 байтов.

Другой неприятной особенностью формата BMP является перевернутый (по отношению к экрану монитора) порядок записи строк в файл — первой записывается самая нижняя строка изображения. Проявляет себя этот недостаток только в том случае, если изображение по высоте превышает размеры экрана, а его просмотр нужно начинать с левого верхнего угла — чтобы добраться до верхней строки, приходится прокручивать файл до самого конца. Использование метода Родена (см. главу 2 «Недокументированные возможности процессоров Intel 80x86») обеспечивает под управлением DOS доступ ко всей оперативной памяти, что позволяет нейтрализовать недостатки формата BMP (большой объем файлов и перевернутый порядок строк).

Формат PCX для 256-цветных изображений

Если необходимо экономить дисковое пространство, то для 256-цветных изображений лучше использовать формат PCX со сжатием по алгоритму Run Length Encoding (RLE). Файл PCX состоит из трех частей: заголовка, сжатого изображения и таблицы палитры. Формат заголовка файла PCX показан в табл. 4.11.

Вообще говоря, данные в файлах PCX могут храниться в несжатом виде — в этом случае байт признака уплотнения данных в заголовке файла имеет значение 0. Неупакованный PCX формат может использоваться, например, для хранения черно-белых фотографий в формате 256 оттенков серого.

Однако в целях сокращения объема файла обычно применяется сжатие по RLE. Способ кодирования RLE предполагает выделение последовательностей пикселов одинакового цвета и запоминание количества повторений в специальном байте-счетчике. Каждая строка изображения кодируется отдельно, причем правила кодирования следующие:

- отдельная точка со значением кода цвета менее C0h будет просто представлена своим кодом;
- отдельная точка со значением кода цвета, которое больше или равно C0h, записывается в виде пары кодов, первым из которых будет байт счетчика со значением C1h, а вторым — собственно байт кода цвета точки;
- горизонтальный отрезок из N (не более 63) одноцветных точек будет представлен парой кодов, первым из которых будет байт счетчика со значением N + C0h, а вторым — байт кода цвета.

Таблица 4.11. Формат заголовка файла PCX

Смещение	Размер	Описание
00h	BYTE	Сигнатура 0Ah — признак файла PCX
01h	BYTE	Версия файла PCX: 0 — версия 2.5; 2 — версия 2.8 с описанием палитры; 3 — версия 2.8 без описания палитры; 5 — версия 3.0
02h	BYTE	Признак уплотнения данных: 0 — сжатие не производилось; 1 — сжатие выполнено методом RLE)
03h	BYTE	Количество цветовых слоев
04h	WORD	Координата X_{min} левого верхнего угла изображения
06h	WORD	Координата Y_{min} левого верхнего угла изображения
08h	WORD	Координата X_{max} правого нижнего угла изображения
0Ah	WORD	Координата Y_{max} правого нижнего угла изображения
0Ch	48 байт	Описание палитры для 16-цветного режима (16 записей, по 3 байта на каждый цвет)

Смещение	Размер	Описание
40h	BYTE	Зарезервировано
41h	BYTE	Число битовых плоскостей
42h	WORD	Число байтов в одной строке изображения (всегда четное)
44h	WORD	Признак типа изображения: 1 — цветное или черно-белое; 2 — с градациями серого
46h	58 байт	Зарезервировано

Если в байте версии файла PCX установлено значение 5, то последние 769 байт файла хранят структуру, содержащую байт — признак формата регистров ЦАП и следующую за ним таблицу палитры (768 байт). Если в байте признака формата регистров записано значение 0Ah, то используется 6-разрядное кодирование основных цветовых компонентов, а если значение 0Ch — 8-разрядное кодирование. Таблицу нужно загрузить в регистры ЦАП видеоконтроллера при помощи подфункции 12h функции 10h прерывания Int 10h в соответствии со значением байта признака формата. Каждая из 256 строк таблицы состоит из трех байтов, содержащих значения компонентов соответствующего цветового оттенка в порядке, используемом функцией загрузки палитры: красный, зеленый, синий.

Палитра PCX в 256-цветном режиме часто не совпадает с используемой по умолчанию палитрой VGA, что создает дополнительные сложности при просмотре изображений — если кроме картинки вы хотите вывести какой-то поясняющий текст, то он поменяет цвет при смене палитры. Поэтому, несмотря на то, что в файле кодирование 256-цветное, при его создании и последующем просмотре удобнее всего использовать графические режимы TrueColor.

При обмене информацией между программами при помощи PCX-файлов возможны ситуации трех основных видов:

- требуется передавать изображение только между своими программами, входящими в один пакет;
- требуется принять изображение, созданное чужой программой;
- нужно передать (транспортировать) изображение из своей программы в чужую.

При обмене файлами изображений внутри одного программного пакета никаких особых сложностей нет: можно использовать любую цветовую палитру. Вообще говоря, для транспортировки данных вы даже можете создать на основе принципа RLE свой собственный

метод сжатия информации, нестандартный, но оптимальный в вашем конкретном случае.

При приеме файлов из чужих программ приходится либо устанавливать ту палитру, которая приходит вместе с файлом (что приводит к искажению цветов в вашей программе), либо производить перекодировку цветов изображения из чужой палитры в свою (что приводит к искажению изображения), либо работать в режиме TrueColor (что связано с проблемами совместимости видеоконтроллеров).

При транспортировке файлов из своего пакета программ в чужой следует учитывать, что многие редакторы изображений корректно работают только с восьмибитовым форматом цветовых компонентов, то есть значение байта признака формата регистров ЦАП должно быть равно 0Ch, а данные в таблице палитры должны соответствовать этому формату. Например, если нужно передать в MS Photo Editor файл, созданный в стандартном 256-цветном VGA-режиме DOS, то после считывания палитры из регистров ЦАП нужно перед записью таблицы в файл сдвинуть в ней каждый байт данных влево на два разряда.

Если требуется реализовать режим фотографии с 256 градациями серого цвета, то соответствующую таблицу палитры нужно вначале сформировать. Генерация таблицы выполняется по примитивному алгоритму: в первую строку записываются три байта со значением ноль, а в каждой следующей строке значения для всех компонент увеличиваются на 1 (получаем в результате в первой строке черный цвет, а в последней — белый).

Ситуация, когда приходится иметь дело с графическим редактором, работающим только с конкретной палитрой, возникает довольно редко. Однако если такая необходимость есть, то приходится заимствовать таблицу палитры вместе с байтом формата ЦАП. Для этого нужно создать файл (для любого произвольного изображения) в чужом пакете, «взять» из него последние 789 байт и либо записать в виде загружаемого двоичного файла, либо перекодировать в текстовый include-файл. Все изображения, которые вы предназначаете для работы с таким редактором, должны создаваться с использованием заимствованной палитры.

Примеры создания, записи и считывания файлов в форматах BMP и PCX будут рассмотрены в главе 6 «Работа с дисками».

Глава 5
Работа с мышью

Если клавиатура является основным средством ввода текста, то мышь — это основное устройство ввода информации и управления системой в графических режимах. К сожалению, очень часто получается так, что в группе совместно работающих устройств половина имеет детальное описание, а другая половина не документирована вообще. Но если нет документации хотя бы на одно устройство, то невозможно использовать всю группу! Например, последовательный порт в настоящее время применяется практически только для подключения мыши, так как для остальных устройств он слишком медленно работает. Режимы работы и регистры последовательного порта описаны во многих источниках, например, в [6, 7, 30], а описание правил работы с мышью MS Mouse найти очень сложно. С мышью типа PS/2 ситуация иная — имеется описание протокола передачи данных, но не описаны команды контроллера, к которому она подключается.

Странно, но факт: сведения о сложных электронных компонентах компьютера (например, микропроцессорах) в Интернете представлены в полном объеме, а информация о простых — отсутствует. В данной главе основное внимание будет уделено той части информации о работе устройств типа «мышь», которая отсутствует в общедоступных источниках.

Функции DOS, предназначенные для работы с мышью

Универсальный драйвер мыши был предложен фирмой Microsoft. Он обеспечивает унифицированный интерфейс для работы с манипуляторами «мышь» или «трекбол» любого типа. Драйвер

позволяет выполнять свыше 40 различных функций. Все фирмы-изготовители манипуляторов делают драйверы для своих устройств совместимыми с драйвером Microsoft, но каждая фирма вносит в них усовершенствования, создавая программистам множество ненужных проблем. Кроме того, стандартные функции мыши в MS-DOS рассчитаны только на текстовые и устаревшие 16-цветные графические режимы: попытка обращения к ним при использовании видеорежимов SVGA, HiColor и TrueColor приводит в лучшем случае к «глюкам» на экране монитора, в худшем — к зависанию системы.

В отличие от устройств других типов, аппаратный интерфейс мыши стандартизирован гораздо лучше, чем программный, и работать с мышью желательно через аппаратуру. Использовать прерывания DOS необходимо только для устройств, не имеющих простого общепринятого интерфейса (например, подключенных через инфракрасный порт или шину USB). Мы не будем подробно рассматривать все функции MS-DOS, предназначенные для обслуживания мыши (см. [3, 10]), поскольку большая часть из них не работает должным образом в современных видеорежимах. Ниже перечислены только те функции, которые не взаимодействуют напрямую с видеоконтроллером и потому особых проблем не вызывают. При этом программист должен сам написать подпрограммы, обеспечивающие отображение курсора мыши и перемещение его по экрану в конкретном видеорежиме.

Прерывание 33h, функция 0000h: проверить наличие драйвера мыши и произвести сброс

Функция проверяет, загружен ли драйвер мыши в память компьютера, и, если драйвер загружен, выполняет общий сброс аппаратного и программного обеспечения мыши.

При вызове функции в регистр AX должен быть помещен код 0.

После выполнения функции в случае отсутствия драйвера в регистре AX будет возвращено значение 0. При наличии драйвера мыши в регистрах будет размещена следующая информация:

- в AX — код FFFFh;
- в BX — код типа мыши (FFFFh — стандартная мышь Microsoft с двумя клавишами, любое другое значение — мышь нестандартная).

Параметры драйвера после сброса следующие:

- координаты курсора установлены на центр экрана;
- для отображения назначена видеостраница 0;
- курсор находится в невидимом состоянии, то есть не отображается на экране;
- курсору придана форма, действующая по умолчанию (в текстовых режимах – форма негативного прямоугольника, в графических режимах – форма стрелки);
- работа пользовательского обработчика сообщений мыши заблокирована;
- эмуляция светового пера разрешена;
- установлена чувствительность мыши к перемещению по горизонтали 8:8 микки на пиксел, по вертикали – 16:8 микки на пиксел;
- порог удвоения скорости равен 64 микки/с;
- область отображения курсора мыши охватывает весь экран.

Из всего перечисленного выше наиболее важным является то, что после сброса курсор на экран не отображается. При работе в текстовом режиме нужно вызвать функцию 0001h, чтобы курсор появился на экране. В современных графических режимах курсор средствами драйвера не отображается или отображается неправильно, поэтому с точки зрения драйвера он должен оставаться скрытым, то есть функцию 0001h вызывать нельзя.

Прерывание 33h, функция 0001h: отобразить курсор мыши на экране

Функция 0001h делает курсор мыши видимым.

При вызове функции в регистр AX должен быть помещен код 0001h.

Никаких выходных параметров функция не имеет.

Применять данную функцию можно только в текстовых режимах.

Прерывание 33h, функция 0002h: убрать курсор мыши с экрана

Функция 0002h делает курсор мыши невидимым.

При вызове функции в регистр AX должен быть помещен код 0002h.

Выходных параметров функция не имеет.

Как в текстовых, так и в графических режимах курсор накладывается поверх изображения, искажая его, поэтому перед выводом курсора драйвер запоминает участок изображения под ним. Функция 0002h осуществляет стирание курсора и восстановление изображения. Данную функцию необходимо использовать перед перерисовкой изображения на экране или при перемещении курсора в другую позицию. Поскольку в современных графических режимах функция 0001h не работает, то не применяется и обратная ей функция 0002h.

Прерывание 33h, функция 0003h: получить информацию о положении курсора и состоянии кнопок мыши

Функция 0003h позволяет определить текущее состояние кнопок мыши и текущее положение курсора.

При вызове функции в регистр AX должен быть помещен код 0003h.

После выполнения функции в регистрах будет возвращена следующая информация:

- в BX — текущее состояние кнопок мыши (бит 0 — состояние левой кнопки, бит 1 — состояние правой кнопки, бит 2 — состояние средней кнопки);
- в CX — горизонтальная координата курсора (X);
- в DX — вертикальная координата курсора (Y).

Если кнопка мыши нажата, значение соответствующего бита регистра BX будет установлено в 1, если отпущена — в 0.

Серьезный недостаток данной функции заключается в том, что невозможно определить, была ли нажата (отпущена) кнопка мыши в текущей позиции курсора, или это произошло ранее в другом месте экрана. Поэтому функция 0003h применяется только в графических видеорежимах — в подпрограмме, отвечающей за перемещение курсора по экрану. Для получения более точной информации о нажатиях и отпусканиях кнопок мыши используются функции 0005h и 0006h.

Прерывание 33h, функция 0004h: установить новое положение курсора

Функция 0004h позволяет изменить положение курсора на экране по команде из программы.

При вызове функции в регистры должна быть занесена следующая информация:

- в AX — код 0004h;
- в CX — горизонтальная координата курсора (X);
- в DX — вертикальная координата курсора (Y).

Выходных параметров функция не имеет.

Положение координаты задается в пикселах относительно левого верхнего угла экрана, причем предполагается, что ось Y направлена сверху вниз.

Прерывание 33h, функция 0005h: получить информацию о нажатиях кнопок мыши

Функция 0005h позволяет получить текущее состояние всех кнопок мыши, а также определить, была ли нажата заданная кнопка, сколько раз ее нажимали с момента последнего опроса, и в какой позиции экрана было осуществлено последнее нажатие.

При вызове функции в регистры должна быть занесена следующая информация:

- в AX — код 0005h;
- в BX — номер кнопки, которую требуется опросить (0 — правая кнопка, 1 — левая кнопка, 2 — средняя кнопка).

После выполнения функции в регистрах будет возвращена следующая информация:

- в AX — текущее состояние кнопок мыши (бит 0 — состояние левой кнопки, бит 1 — состояние правой кнопки, бит 2 — состояние средней кнопки);
- в BX — число нажатий на указанную кнопку с момента последнего вызова данной функции для данной кнопки;
- в CX — горизонтальная координата курсора (X) в момент последнего нажатия указанной кнопки;
- в DX — вертикальная координата курсора (Y) в момент последнего нажатия указанной кнопки.

Прерывание 33h, функция 0006h: получить информацию об отпусканиях кнопок мыши

Функция 0006h позволяет получить текущее состояние всех кнопок мыши, а также определить, имело ли место отпускание заданной

кнопки, сколько раз ее отпускали с момента последнего опроса, и в какой позиции экрана было осуществлено последнее отпускание.

При вызове функции в регистры должна быть занесена следующая информация:

- в AX — код 0006h;
- в BX — номер кнопки, которую требуется опросить (0 — правая кнопка, 1 — левая кнопка, 2 — средняя кнопка).

После выполнения функции в регистрах будет возвращена следующая информация:

- в AX — текущее состояние кнопок мыши (бит 0 — состояние левой кнопки, бит 1 — состояние правой кнопки, бит 2 — состояние средней кнопки);
- в BX — число отпусканий указанной кнопки с момента последнего вызова данной функции для данной кнопки;
- в CX — горизонтальная координата курсора (X) в момент последнего отпускания указанной кнопки;
- в DX — вертикальная координата курсора (Y) в момент последнего отпускания указанной кнопки.

Прерывание 33h, функция 0007h: задать горизонтальный диапазон перемещения курсора

Функция 0007h служит для ограничения пределов горизонтального перемещения курсора.

При вызове функции в регистры должна быть занесена следующая информация:

- в AX — код 0007h;
- в CX — координата левой границы перемещения курсора (X_{min});
- в DX — координата правой границы перемещения курсора (X_{max}).

Выходных параметров функция не имеет.

Функция 0007h используется совместно с функцией для задания на экране рабочего окна, за пределы которого не должен выходить курсор мыши. В простейшем случае функция применяется, чтобы предотвратить уход части изображения графического курсора за границу экрана.

Прерывание 33h, функция 0008h: задать вертикальный диапазон перемещения курсора

Функция 0008h служит для ограничения пределов вертикального перемещения курсора.

При вызове функции в регистры должна быть занесена следующая информация:

- в AX — код 0008h;
- в CX — координата верхней границы перемещения курсора (Y_{min});
- в DX — координата нижней границы перемещения курсора (Y_{max}).

Выходных параметров функция не имеет.

Как и функция 0007h, данная функция чаще всего применяется, чтобы предотвратить уход части изображения курсора за границу экрана.

Прерывание 33h, функция 000Ch: задать подпрограмму пользователя обработчику прерывания мыши

Функция 000Ch позволяет пользователю установить собственный обработчик прерывания мыши.

При вызове функции в регистры должна быть занесена следующая информация:

- в AX — код 000Ch;
- в CX — маска условий вызова;
- в ES:DI — дальний указатель на подключаемую подпрограмму пользователя.

Выходных параметров функция не имеет.

При вызове подпрограммы драйвер передает ей параметры через регистры. В регистрах будут размещены следующие значения:

- в AX — маска условий вызова;
- в BX — текущее состояние кнопок мыши (бит 0 — состояние левой кнопки, бит 1 — состояние правой кнопки, бит 2 — состояние средней кнопки);
- в CX — горизонтальная координата курсора (X);
- в DX — вертикальная координата курсора (Y);

- в SI — горизонтальный отсчет (микки);
- в DI — вертикальный отсчет (микки);
- в DS — сегмент данных драйвера мыши.

Разряды маски условий вызова имеют следующее назначение (если разряд установлен в 1, то выполняется соответствующее действие):

- бит 0 — вызывать подпрограмму пользователя в случае перемещения мыши;
- бит 1 — вызывать подпрограмму пользователя при нажатии левой кнопки;
- бит 2 — вызывать подпрограмму пользователя при отпускании левой кнопки;
- бит 3 — вызывать подпрограмму пользователя при нажатии правой кнопки;
- бит 4 — вызывать подпрограмму пользователя при отпускании правой кнопки;
- бит 5 — вызывать подпрограмму пользователя при нажатии средней кнопки;
- бит 6 — вызывать подпрограмму пользователя при отпускании средней кнопки;
- биты 7–15 — зарезервированы.

Функция 000Ch очень полезна при работе в современных графических режимах — подпрограмма вывода курсора, подключенная к драйверу, может быть настроена на особенности конкретного видеорежима и даже на конкретный тип видеоконтроллера. Следовательно, при выводе курсора не будет возникать никаких побочных искажений изображения.

Прерывание 33h, функция 000Fh: изменить чувствительность мыши к перемещению

Функция 000Fh позволяет управлять чувствительностью мыши.

При вызове функции в регистры должна быть занесена следующая информация:

- в AX — код 000Fh;
- в CX — число микки на 8 пикселов по горизонтали (по умолчанию 8);
- в DX — число микки на 8 пикселов по вертикали (по умолчанию 16).

Выходных параметров функция не имеет

Прерывание 33h, функция 0013h: задать порог удвоения скорости

Функция 0013h позволяет изменить порог удвоения скорости перемещения мыши. Если скорость перемещения мыши превышает заданный порог, то скорость перемещения курсора по экрану удваивается.

При вызове функции в регистры должна быть занесена следующая информация:

- в AX — код 0013h;
- в DX — пороговая скорость в микки/с (по умолчанию равна 64 микки/с).

Выходных параметров функция не имеет.

Работа с мышью через последовательный порт

Существует несколько фактических стандартов на способы подключения координатных устройств к компьютеру. Эти стандарты предусматривают различные способы подключения устройства и различные форматы передачи данных. В настоящее время применяются два основных способа подключения мыши к персональному компьютеру — подключение через последовательный порт (Serial Mouse) и подключение через разъем дополнительного устройства PS/2. Другие способы, например подключение через шину USB или инфракрасный порт, пока что мало распространены и практически не документированы. Изучение приемов работы на аппаратном уровне с манипуляторами типа мышь начнем с самой распространенной разновидности — Serial Mouse.

Форматы передачи данных Serial Mouse

Внутренняя структура драйвера мыши определяется в первую очередь используемым мышью форматом передачи данных. Для устройств, подключаемых через последовательный порт, применяется ряд различных форматов: группа форматов, базирующихся на протоколе MS Mouse, и формат PC Mouse [35, 55, 58].

Группа форматов Microsoft Mouse в настоящее время стала основной для координатных устройств, подключаемых к последовательному порту, вытеснив из этой области другие виды протоколов. Все

форматы этой группы являются расширениями 7-битного формата данных фирмы Microsoft, приведенного в табл. 5.1. Обозначения в таблице расшифровываются следующим образом:

- X0–X7 — перемещение по оси X (целое число со знаком);
- Y0–Y7 — перемещение по оси Y (целое число со знаком);
- L — состояние левой кнопки (0 — отпущена, 1 — нажата);
- R — состояние правой кнопки (0 — отпущена, 1 — нажата).

Таблица 5.1. Стандартный формат Microsoft (MS Mouse)

Номер байта в посылке	Номер бита						
	6	5	4	3	2	1	0
1	1	L	R	Y7	Y6	X7	X6
2	0	X5	X4	X3	X2	X1	X0
3	0	Y5	Y4	Y3	Y2	Y1	Y0

Данный формат был введен для двухкнопочной мыши. Средняя кнопка трехкнопочной мыши, поддерживающей работу с несколькими протоколами, при работе в режиме MS Mouse эквивалентна правой.

Старший бит посылки (бит 6) используется для самоконтроля и синхронизации: признаком начала передачи очередного пакета из трех байт служит единица в этом бите. Программа — обработчик прерывания мыши должна удостовериться, что следующие два байта данных имеют в шестом разряде нулевое значение. В противном случае произошел сбой в процессе передачи и следует проигнорировать принятый пакет.

Скорость приема-передачи данных принята равной 1200 бод, длина передаваемого слова — 7 бит, контроль по четности не используется, число стоповых битов равно 1 (прежде чем начинать работу с мышью, нужно загрузить эти значения в регистры последовательного порта, к которому она подключена). Передача данных производится только в том случае, если изменяется состояние кнопок мыши или координат X и Y. Ось Y в режиме MS Mouse направлена сверху вниз, как у дисплея.

Для обеспечения нормальной работы с трехкнопочными устройствами протокол Microsoft пришлось дополнить четвертым байтом, который служит одной-единственной цели — обеспечивает передачу состояния средней кнопки мыши (в пятом разряде, обозначенном

символом M). Обязательно нужно учитывать, что передача пакета из четырех слов выполняется только в случае изменения состояния средней кнопки, а в остальных случаях передаются первые три слова. Данный протокол, получивший название Microsoft Plus, показан в табл. 5.2.

Таблица 5.2. Формат Microsoft Plus (M+) для трехкнопочной мыши

Номер байта в посылке	Номер бита						
	6	5	4	3	2	1	0
1	1	L	R	Y7	Y6	X7	X6
2	0	X5	X4	X3	X2	X1	X0
3	0	Y5	Y4	Y3	Y2	Y1	Y0
4[1]	0	M	0	0	0	0	0

[1] Передача четвертого байта производится только в случае изменения состояния средней кнопки мыши.

Конкуренция между изготовителями компьютерного оборудования приводит к тому, что на рынке появляется все больше «навороченных» устройств, снабженных рядом дополнительных функций. Дополнительные возможности требуют, естественно, передачи дополнительной информации от устройства к компьютеру. Например, формат данных трехкоординатной (3D) мыши, показанный в табл. 5.3, включает координату Z (Z0–Z3, целое число со знаком) и состояние специальной кнопки R_0.

Таблица 5.3. Формат 3D Serial mouse

Номер байта в посылке	Номер бита						
	6	5	4	3	2	1	0
1	1	L	R	Y7	Y6	X7	X6
2	0	X5	X4	X3	X2	X1	X0
3	0	Y5	Y4	Y3	Y2	Y1	Y0
4	0	0	R0	Z3	Z2	Z1	Z0

Существует группа универсальных устройств, которые поддерживают два типа протоколов — PC Mouse и MS Mouse. Переключение между протоколами осуществляется аппаратно (переключателем на корпусе мыши) или программно (путем подачи специального

сигнала со стороны драйвера мыши). Особенностью устройств данного типа является 8-битная длина слова данных во всех режимах работы (при этом в режиме MS Mouse шестой разряд в каждом слове данных просто дублируется в седьмой, как показано в табл. 5.4). Соответственно при использовании подобных устройств для последовательного порта необходимо установить следующие параметры работы: скорость приема-передачи данных 1200 бод, длина передаваемого слова — 8 бит, контроль по четности не используется, число стоповых бит равно 1. Однако даже если задана длина передаваемого слова 7 бит, в режиме MS Mouse седьмой разряд игнорируется, и нарушения процесса передачи не происходит. Таким образом, устройства данного типа могут успешно взаимодействовать с любым драйвером, поддерживающим стандартный протокол Microsoft.

Таблица 5.4. Нестандартный 8-битный типа MS Mouse для мыши с двумя альтернативными протоколами передачи данных

Номер байта в посылке	Номер бита							
	7	6	5	4	3	2	1	0
1	1	1	L	R	Y7	Y6	X7	X6
2	0	0	X5	X4	X3	X2	X1	X0
3	0	0	Y5	Y4	Y3	Y2	Y1	Y0

Предложенный фирмой IBM формат Mouse System (PC Mouse), который показан в табл. 5.5, в настоящее время почти не применяется, однако до сих пор поддерживается некоторыми универсальными устройствами. Обозначения в таблице расшифровываются следующим образом:

- X0–X7 — перемещение по оси X;
- Y0–Y7 — перемещение по оси Y;
- L — состояние левой кнопки (0 — нажата, 1 — отпущена);
- M — состояние средней кнопки (0 — нажата, 1 — отпущена);
- R — состояние правой кнопки (0 — нажата, 1 — отпущена).

Особенность данного формата состоит в том, что для определения перемещения по оси X нужно сложить значения X' и X" (байты 2 и 4), а для определения перемещения по оси Y — значения Y' и Y" (байты 3 и 5). Столь хитрый способ передачи координаты предназначен для обеспечения уникальности признака начала кадра

(единица в бите 7 и нули в битах 3–6 первого байта посылки). Ось Y мыши в данном формате направлена вверх, то есть противоположно оси Y дисплея.

Таблица 5.5. Формат Mouse System (PC Mouse)

Номер байта в посылке	Номер бита							
	7	6	5	4	3	2	1	0
1	1	0	0	0	0	L	M	R
2	X7'	X6'	X5'	X4'	X3'	X2'	X1'	X0'
3	Y7'	Y6'	Y5'	Y4'	Y3'	Y2'	Y1'	Y0'
4	X7"	X6"	X5"	X4"	X3"	X2"	X1"	X0"
5	Y7"	Y6"	Y5"	Y4"	Y3"	Y2"	Y1"	Y0"

Для работы с устройством типа PC Mouse необходимо установить следующие параметры последовательного порта: скорость приема-передачи 1200 бод, длина передаваемого слова 8 бит, контроль по четности не используется, число стоповых бит равно 1. Передача данных производится только в том случае, если изменяется состояние кнопок или координат X и Y.

Программирование порта последовательной передачи данных

Для того чтобы можно было принимать и обрабатывать информацию, передаваемую мышью, необходимо запрограммировать регистры последовательного порта в соответствии с параметрами протокола передачи данных, который она использует.

Последовательный порт передает и принимает информацию в асинхронном режиме. Формат передачи данных последовательного порта в обобщенном виде представлен на рис. 5.1.

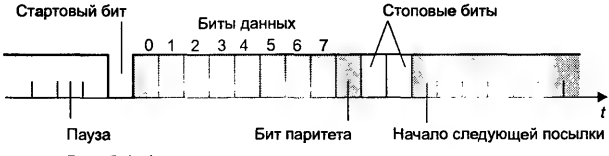


Рис. 5.1. Формат передачи данных последовательного порта

Передача данных начинается посылкой стартового бита, за которым следует от 5 до 8 бит данных. Если используется контроль по четности, то за битами данных следует бит паритета. Завершает посылку стоповый сигнал длительностью в 1, 1,5 или 2 тактовых интервала.

Мышь может быть подключена к одному из двух последовательных портов IBM-совместимого компьютера: COM1 или COM2. Порт COM1 имеет базовый адрес 3F8h и занимает в пространстве ввода-вывода диапазон адресов 3F8h–3FFh. Он может вырабатывать прерывание IRQ4. Базовый адрес порта COM2 — 2F8h, диапазон адресов — 2F8h–2FFh, вырабатываемое прерывание — IRQ3. Чтобы избежать повторений, в дальнейшем для адресов регистров будут использоваться обозначения вида xF...h, где x соответствует 3 для порта COM1 и 2 для порта COM2.

Адрес xF8h разделяют три регистра: *регистр данных передатчика* (THR), доступный только для записи, *регистр данных приемника* (RBR), доступный только для считывания, и *регистр младшего байта делителя частоты* (DLL).

Адрес xF9h разделяют два регистра: *регистр управления прерываниями* (IER) и *регистр старшего байта делителя частоты* (DLM).

Формат регистра управления прерываниями показан на рис. 5.2. Разряды этого регистра имеют следующее назначение:

- бит 0 — прерывание при поступлении байта данных (0 — запрещено, 1 — разрешено);
- бит 1 — прерывание при завершении передачи байта данных (0 — запрещено, 1 — разрешено);
- бит 2 — прерывание по ошибке или обрыву линии (0 — запрещено, 1 — разрешено);
- бит 3 — прерывание по сигналу от модема (0 — запрещено, 1 — разрешено);
- биты 4–7 — не используются и должны быть установлены в 0.

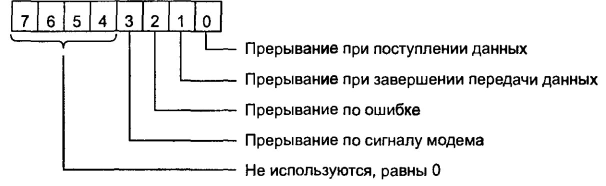


Рис. 5.2. Формат регистра управления прерываниями

Регистр идентификации прерывания (IIR) расположен по адресу xFAh. Формат регистра показан на рис. 5.3. Разряды регистра имеют следующее назначение:

- бит 0 — запрос прерывания (0 — нет запроса, 1 — есть запрос);
- биты 1 и 2 идентифицируют источник сигнала прерывания:
  - 00 — изменение состояния модема;
  - 01 — завершение передачи байта;
  - 10 — завершение приема байта;
  - 11 — ошибка при приеме данных или обрыв линии;
- бит 3 — прерывание по тайм-ауту;
- биты 4 и 5 — не используются (установлены в 0);
- биты 6 и 7 — признак режима FIFO:
  - 00 — обычный режим;
  - 10 — режим FIFO 16550;
  - 11 — режим FIFO 16550A.



Рис. 5.3. Формет регистра идентификеции прерывений

Регистр управления линией (LCR) находится по адресу xFBh. Формат регистра приведен на рис. 5.4. Назначение разрядов регистра следующее:

- биты 0 и 1 — длина передаваемого слова:
  - 00 — 5 бит;
  - 01 — 6 бит;
  - 10 — 7 бит;
  - 11 — 8 бит;
- бит 2 — число стоповых битов: (0 — один бит, 1 — два бита);
- бит 3 — наличие контроля по четности (0 — бит паритета отсутствует, 1 — присутствует);

- бит 4 — тип контроля (0 — контроль на нечетность, 1 — на четность);
- бит 5 — имитация контроля четности (0 — нормальный контроль паритета, 1 — имитация контроля — контрольный бит генерируется на основании полученного сигнала);
- бит 6 — формирование сигнала «Обрыв линии»;
- бит 7 — управление регистрами xF8h и xF9h (при установке в 0 этого бита данные регистры используются как регистр данных и регистр управления прерываниями соответственно, а при установке 1 — для загрузки делителя частоты тактового генератора).



Рис. 5.4. Формат рвгистра управления линивй

Скорость приема-передачи задается делением частоты встроенного тактового генератора (115 200 Гц) на значение, записываемое в регистры xF8h и xF9h. Перед записью делителя необходимо переключить эти регистры в соответствующий режим, установив в единицу бит 7 регистра управления линией xFBh. Для установки используемой мышью скорости 1200 бод необходимо записать число 96 (60h) в регистр младшего байта делителя частоты и 0 — в регистр старшего байта. После записи делителя нужно вернуть регистры в исходное состояние, сбросив в ноль бит 7 регистра управления.

Регистр управления модемом (MCR) расположен по адресу xFCh. Формат регистра показан на рис. 5.5. Разряды регистра имеют следующее назначение:

- бит 0 — управление сигналом готовности выходных данных DTR;
- бит 1 — управление сигналом готовности к приему данных RTS;
- бит 2 — управление выходным сигналом OUT1;
- бит 3 — управление выходным сигналом OUT2;

- бит 4 — запуск самотестирования контроллера последовательного порта (0 — нормальный режим, 1 — режим самодиагностики);
- биты 5–7 — зарезервированы и должны быть установлены в 0.

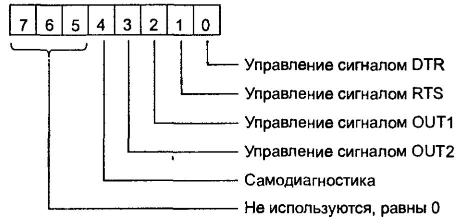


Рис. 5.5. Формат регистра управления модемом

Регистр состояния приемопередатчика (LSR) находится по адресу xFDh. Формат регистра показан на рис. 5.6. Назначение разрядов следующее:

- бит 0 — готовность данных (устанавливается в 1, когда данные получены приемником и готовы для считывания);
- бит 1 — переполнение (устанавливается в 1 при ошибке переполнения приемника);
- бит 2 — ошибка паритета (устанавливается в 1 при обнаружении ошибки по четности);
- бит 3 — ошибка кадра (устанавливается в 1 при сбое синхронизации);
- бит 4 — обнаружение сигнала «Обрыв линии» (устанавливается в 1, если сигнал на входе приемника длительное время находится в состоянии 0);
- бит 5 — регистр данных передатчика пуст (устанавливается в 1, когда завершена передача байта данных и в регистр можно записывать следующий байт);
- бит 6 — регистр сдвига передатчика пуст (устанавливается в 1, когда свободны все регистры передатчика);
- бит 7 — ошибка в буфере FIFO (устанавливается в 1, если буфер содержит хотя бы один байт, принятый с ошибкой).

Разряды 1–4 после считывания информации из регистра состояния сбрасываются. Разряд 0 сбрасывается при чтении данных из регистра xF8h.



Рис. 5.6. Формат регистра состояния приемопередатчика

Регистр состояния модема (MSR) находится по адресу xFEh. Формат регистра показан на рис. 5.7. Разряды регистра имеют следующее назначение:

- бит 0 — изменение состояния линии CTS;
- бит 1 — изменение состояния линии DSR;
- бит 2 — изменение состояния линии RI;
- бит 3 — изменение состояния линии DCD;
- бит 4 — состояние линии CTS;
- бит 5 — состояние линии DSR;
- бит 6 — состояние линии RI;
- бит 7 — состояние линии DCD.

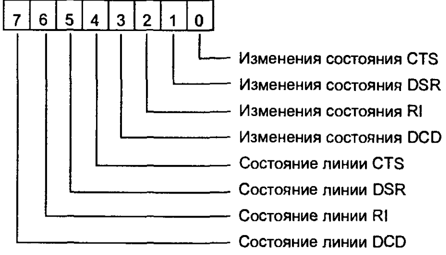


Рис. 5.7. Формат регистра состояния модема

Признаки изменения состояния сигналов на линиях (биты 0–3) сбрасываются после считывания информации из регистра.

Непосредственная работа с мышью типа MS Mouse

Рассмотрим непосредственную работу с мышью на примере манипуляторов, использующих стандартный формат MS Mouse.

Процесс установки нового драйвера мыши включает целый ряд операций, которые необходимо выполнять в следующем порядке.

1. Запретить прерывания от последовательного порта.
2. Загрузить в регистры COM-порта параметры, соответствующие протоколу передачи данных.
3. Установить вектор прерывания на новую программу-обработчик.
4. Разрешить аппаратуре контроллера прерываний формировать прерывания от используемого мышью COM-порта.
5. Подать на мышь напряжение питания.
6. Разрешить генерацию прерываний COM-портом.

Разрешить или запретить прохождение прерываний от последовательного порта можно при помощи регистра маски ведущей микросхемы контроллера прерываний, к которой присоединены линии запросов от COM1 (линия IRQ4) и COM2 (линия IRQ3). Работа с контроллером прерываний уже была описана в разделе «Контроллер прерываний» главы 1 «Работа с клавиатурой». После завершения обработки прерывания необходимо разблокировать контроллер, послав ему команду завершения обработки прерывания EOI.

Настройка вектора прерывания заключается в записи начального адреса программы-обработчика в соответствующие вектору ячейки оперативной памяти. Для записи вектора необходимо занести в сегментный регистр данных нулевое значение, в индексный регистр поместить номер вектора прерывания, помноженный на 4, записать в ячейку памяти (используя косвенную адресацию) слово — смещение, увеличить значение индексного регистра на 2 и записать слово — сегмент адреса обработчика прерывания.

Рассмотрим более подробно последние два этапа, поскольку они имеют ряд неочевидных особенностей. В процессе разработки способа подключения мыши к последовательным портам был использован радиолюбительский трюк: питание мыши осуществляется от сигнальных линий DTR и RTS. Для управления генераций прерываний применяется линия OUT2. Таким образом, чтобы подать питание на мышь и разрешить порту вырабатывать прерывания, необходимо установить в 1 биты 0, 1 и 3 регистра управления модемом, то есть записать в регистр xFCh значение 0Bh.

Чаще всего мышь подключают к порту COM1. Однако если заранее не известно, к какому порту подключена мышь (и подключена ли вообще), то можно выполнить операцию идентификации мыши. Обычно процедура идентификации начинается с проверки того, подключено ли к порту какое-либо устройство. Вообще говоря, после подачи питания любое устройство, подключенное к последовательному порту, должно выдать сигнал готовности к работе DSR, значение которого можно прочитать в регистре состояния модема xFEh. Однако для мыши такой метод неприменим, поскольку сигнал DSR обычно отключают с целью сокращения количества проводников в соединительном кабеле мыши.

Чтобы убедиться, что к порту подключена мышь, поддерживающая протокол MS Mouse, нужно временно отключить ее питание (сбросив сигналы DTR и RTS), подождать 0,2 с (4–5 тиков системных часов) и подать питание вновь. После этого мышь должна выдать код 4Dh (латинская заглавная буква «M» в кодировке ASCII) или цепочку символов, начинающуюся с этого кода. У трехкнопочной мыши идентификационный код состоит не менее чем из двух символов (начинается с буквы M и следующей за ней цифры 3). Идентификационные коды для основных изготовителей оборудования можно при необходимости найти в спецификации PC 99 System Design Guide [80], однако на рынке постоянно появляются новые игроки.

После того как завершено конфигурирование системы (определен используемый мышью порт) и произведена установка обработчика прерываний, новый драйвер мыши начинает работать. Драйвер должен принимать пакеты данных от мыши, вычислять координаты курсора и управлять перемещением курсора по экрану, то есть выполнять те же операции, что и стандартный драйвер Microsoft Mouse. Однако, в отличие от стандартного, «самодельный» драйвер можно настраивать на работу с любым видеоконтроллером в любом возможном видеорежиме. Кроме того, стандартный драйвер работает только в реальном режиме DOS, а собственные драйверы при необходимости можно написать для любого режима работы процессора x86, например, для защищенного или виртуального.

Процедуры, предназначенные для непосредственной работы с мышью на уровне регистров последовательного порта, собраны в листинге 5.1. Поиск мыши по последовательным портам COM1 и COM2 выполняется процедурой MSMouseSearch. После определения порта, к которому подключена мышь, следует запустить подпрограмму SetMSMouseInterrupt, которая настраивает обработчик прерывания (устанавливая параметры порта и соответствующий вектор прерывания),

а затем активизирует порт (подавая на мышь напряжение питания и разблокируя прерывания). Прерывания обрабатываются процедурой MSMouseInterrupt, принимающей от мыши пакеты данных и вычисляющей приращения координат курсора. После завершения работы с мышью необходимо восстановить старый вектор прерывания, вызвав процедуру RestoreOldMSMouseInterrupt.

Листинг 5.1. Универсальные процедуры для работы с мышью MS Mouse

```
DATASEG
; Предыдущее значение системного таймера
Time DD ?
; Номер порта (0 - COM1, 1 - COM2,
; 2 - мышь не обнаружена)
COMPortNum DW 0
; Базовый адрес порта мыши
COMPortBaseAddr DW 3F8h
; Область сохранения старого вектора прерывания
OldMSMouseInterruptOffset  DW ?
OldMSMouseInterruptSegment DW ?
; Номер принимаемого от мыши байта
MouseByteNumber DB 0
; Трехбайтовая структура данных, передаваемая мышью
FirstByte       DB 0
SecondByte      DB 0
ThirdByte       DB 0
; Текущее состояние кнопок
ButtonsStatus   DB 0
; Текущие координаты курсора мыши
XCoordinate     DW 0
YCoordinate     DW 0
; Предыдущая позиция курсора мыши
OldXCoordinate  DW 0
OldYCoordinate  DW 0
; Состояние курсора мыши
; (0 - не отображается, 1 - отображается)
MouseCursorStatus DB 0
ENDS

CODESEG
;****************************************************
;*      ПОИСК МЫШИ ПО ПОСЛЕДОВАТЕЛЬНЫМ ПОРТАМ       *
;* Процедура подготавливает глобальные переменные *
;* COMPortNum и COMPortBaseAddr для подпрограммы  *
;* установки обработчика прерывания               *
;****************************************************
PROC MSMouseSearch NEAR
        pusha
```

продолжение ⇩

Листинг 5.1 (*продолжение*)

```
        push    ES
; Запретить прерывание COM1 и COM2
        cli
        in      AL,21h  ;прочитать маску прерываний
        or      AL,18h  ;запретить IRQ3 и IRQ4
        out     21h,AL  ;заменить маску
        sti
; Настроить ES на сегмент данных BIOS
; для работы с системным таймером
        mov     AX,0
        mov     ES,AX

; ПОИСК МЫШИ ЧЕРЕЗ COM-ПОРТЫ
@@MouseSearch:
        ; Устанавливаем скорость
        ; приема/передачи 1200 бод
        mov     DX,[COMPortBaseAddr]
        add     DX,3
        in      AL,DX
        or      AL,80h  ;установить бит DLAB
        out     DX,AL
        mov     DX,[COMPortBaseAddr]
        mov     AL,60h  ;1200 бод
        out     DX,AL
        inc     DX
        mov     AL,0
        out     DX,AL
        ; Установить длину слова 7 бит, 1 стоповый бит,
        ; четность не контролировать
        mov     DX,[COMPortBaseAddr]
        add     DX,3
        mov     AL,00000010b
        out     DX,AL
        ; Запретить все прерывания
        mov     DX,[COMPortBaseAddr]
        inc     DX
        mov     AL,0
        out     DX,AL

; Проверить, что устройство подключено и является
; мышью типа MSMouse
        ; Запомнить текущее время
        mov     EAX,[ES:046Ch]
        mov     [Time],EAX
        ; Отключить питание мыши и прерывания
        mov     DX,[COMPortBaseAddr]
        add     DX,4    ;регистр управления модемом
        mov     AL,0    ;сбросить DTR, RTS и OUT2
        out     DX,AL
```

```
        ; Ожидать 5 "тиков" (0,2 с)
@@dT:   mov     EAX,[ES:046Ch]
        sub     EAX,[Time]
        cmp     EAX,5
        jb      @@dT
        ; Включить питание мыши
        mov     AL,11b  ;установить DTR и RTS
        out     DX,AL
        ; Очистить регистр данных
        mov     DX,[COMPortBaseAddr]
        in      AL,DX
; Цикл опроса порта
@@WaitData:
        ; Ожидать еще 10 "тиков"
        mov     EAX,[ES:046Ch]
        sub     EAX,[Time]
        cmp     EAX,5+10
        jae     @@NoMouse
        ; Проверить наличие идентификационного байта
        mov     DX,[COMPortBaseAddr]
        add     DX,5
        in      AL,DX
        test    AL,1   ;Данные готовы?
        jz      @@WaitData
        ; Ввести данные
        mov     DX,[COMPortBaseAddr]
        in      AL,DX
        ; Устройство является мышью?
        cmp     AL,'M'
        je      @@End
@@NoMouse:
        inc     [COMPortNum]
        cmp     [COMPortNum],1
        ja      @@End
        sub     [COMPortBaseAddr],100h
        jmp     @@MouseSearch

@@End:  pop     ES
        popa
        ret
ENDP MSMouseSearch

;****************************************************
;* УСТАНОВИТЬ ВЕКТОР ПРЕРЫВАНИЯ ОТ ПОСЛЕДОВАТЕЛЬНОГО *
;*           ПОРТА НА НОВЫЙ ДРАЙВЕР МЫШИ            *
;****************************************************
PROC SetMSMouseInterrupt NEAR
        pusha
        push    DS
        push    ES
```

продолжение

Листинг 5.1 (*продолжение*)

```
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
; Запретить прерывание от COM-порта
        ; Вычислить маскируемый разряд
        mov     CL,[byte ptr COMPortNum]
        mov     CH,10h
        shr     CH,CL   ;маскируемый разряд - в CH
        cli
        in      AL,21h  ;IMR 1-го контроллера прерываний
        or      AL,CH   ;запретить прерывание
        out     21h,AL
        sti
; Обнулить счетчик байтов
        mov     [MouseByteNumber],0
; Установка вектора прерывания на обработчик мыши
        ; Настроить ES на область векторов
        mov     AX,0
        mov     ES,AX
        ; Вычислить адрес вектора
        mov     BX,0Ch
        sub     BX,[COMPortNum]
        shl     BX,2
        ; Сохранить старый вектор
        mov     AX,[ES:BX]
        mov     [OldMSMouseInterruptOffset],AX
        mov     AX,[ES:BX+2]
        mov     [OldMSMouseInterruptSegment],AX
        ; Установить новый вектор
        cli
        mov     AX,offset MSMouseInterrupt
        mov     [ES:BX],AX
        mov     AX,CS
        mov     [ES:BX+2],AX
        sti

; АКТИВИЗИРОВАТЬ ПОРТ
; Разрешаем прерывание по приему данных
        mov     DX,[COMPortBaseAddr]
        inc     DX              ;xF9h
        mov     AL,1
        out     DX,AL
; Установить DTR, RTS и OUT2
; (подать напряжение питания и разрешить прерывания)
        mov     DX,[COMPortBaseAddr]
        add     DX,4            ;xFCh
        mov     AL,00001011b
        out     DX,AL
; Очистить регистр данных
        mov     DX,[COMPortBaseAddr]    ;xF8h
```

```
        in      AL,DX
; Разрешить прерывания от COM1
        not     CH
        cli
        in      AL,21h ;IMR 1-го контроллера прерываний
        and     AL,CH  ;разрешить прерывания
        out     21h,AL
        sti
        pop     ES
        pop     DS
        popa
        ret
ENDP SetMSMouseInterrupt

;*********************************************
;* ВОССТАНОВИТЬ СТАРЫЙ ВЕКТОР ПРЕРЫВАНИЯ МЫШИ *
;*********************************************
PROC RestoreOldMSMouseInterrupt NEAR
        pusha
        push    ES
        ; Настроить ES на область векторов
        mov     AX,0
        mov     ES,AX
        ; Вычислить адрес вектора
        mov     BX,0Ch
        sub     BX,[COMPortNum]
        shl     BX,2
        ; Восстановить старый вектор
        cli
        mov     AX,[OldMSMouseInterruptOffset]
        mov     [ES:BX],AX
        mov     AX,[OldMSMouseInterruptSegment]
        mov     [ES:BX+2],AX
        sti
        pop     ES
        popa
        ret
ENDP RestoreOldMSMouseInterrupt

;*************************************
;* НОВЫЙ ОБРАБОТЧИК ПРЕРЫВАНИЯ ОТ МЫШИ *
;*************************************
PROC MSMouseInterrupt NEAR
        pusha
        push    DS
        sti     ;разрешить маскируемые прерывания
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX

; Проверить наличие данных
        mov     DX,[COMPortBaseAddr]
```

продолжение

Листинг 5.1 (*продолжение*)

```
        add     DX,5          ;xFDh
        in      AL,DX
        test    AL,1   ;Данные готовы?
        jz      @@Error
; Ввести данные
        mov     DX,[COMPortBaseAddr]   ;xFBh
        in      AL,DX
; Разблокировать контроллер прерываний
        mov     AH,AL  ;запомнить данные в AH
        mov     AL,20h
        out     20h,AL ;послать команду EOI
        mov     AL,AH  ;восстановить данные в AL
; Сбросить старший незначащий бит
        and    AL,01111111b

; Определить порядковый номер принимаемого байта
        cmp     [MouseByteNumber],0
        je      @@FirstByte
        cmp     [MouseByteNumber],1
        je      @@SecondByte
        cmp     [MouseByteNumber],2
        je      @@ThirdByte
        jmp     @@Error

; Сохранить первый байт данных
@@FirstByte:
        test    AL,1000000b       ;Первый байт посылки?
        jz      @@Error
        mov     [FirstByte],AL
        inc     [MouseByteNumber] ;увеличить счетчик
        jmp     @@EndMouseInterrupt
; Сохранить второй байт данных
@@SecondByte:
        test    AL,1000000b
        jnz     @@Error
        mov     [SecondByte],AL
        inc     [MouseByteNumber] ;увеличить счетчик
        jmp     @@EndMouseInterrupt
; Сохранить третий байт данных
@@ThirdByte:
        test    AL,1000000b
        jnz     @@Error
        mov     [ThirdByte],AL    ;увеличить счетчик
        mov     [MouseByteNumber],0
; (Пакет данных от мыши принят полностью).

; Записать новое значение состояния кнопок мыши
        mov     AL,[FirstByte]
```

```
        shr     AL,4
        mov     [ButtonsStatus],AL

; Прибавить перемещение по X к координате X
        mov     AL,[FirstByte]
        shl     AL,6
        or      AL,[SecondByte]
        cbw
        add     AX,[XCoordinate]
        ; Курсор не должен выходить за левую или
        ; правую границу экрана
        js      @@X1
        cmp     AX,ScreenLength
        jb      @@X2
        ; Установить координату X по правой границе
        mov     AX,ScreenLength-1
        jmp     @@X2
@@X1:   ; Установить координату X по левой границе
        xor     AX,AX
@@X2:   mov     [XCoordinate],AX

; Прибавить перемещение по Y к координате Y
        mov     AL,[FirstByte]
        and     AL,00001100b
        shl     AL,4
        or      AL,[ThirdByte]
        cbw
        add     AX,[YCoordinate]
        ; Курсор не должен выходить за верхнюю или
        ; нижнюю границу экрана
        js      @@Y1
        cmp     AX,ScreenHeigth
        jb      @@Y2
        ; Установить координату X по нижней границе
        mov     AX,ScreenHeigth-1
        jmp     @@Y2
@@Y1:   ; Установить координату X по верхней границе
        xor     AX,AX
@@Y2:   mov     [YCoordinate],AX

; Показать курсор в новой позиции
        cmp     [MouseCursorStatus],0
        je      @@EndMouseInterrupt
        call    ShowNewMouseCursorPosition
        jmp short @@EndMouseInterrupt

@@Error:
; Произошел сбой в порядке передачи информации от
; мыши, обнулить счетчик байтов пакета данных
        mov     [MouseByteNumber],0
```

продолжение ⇒

Листинг 5.1 (*продолжение*)

```
@@EndMouseInterrupt:
        pop     DS
        popa
        iret
ENDP MSMouseInterrupt
ENDS
```

Приведенная в листинге 5.2 программа MSMouseMain, демонстрирующая работу с мышью через последовательный порт, использует процедуры из листинга 5.1. Для вывода курсора на экран вызывается процедура ShowNewMouseCursorPosition, которая инвертирует байт атрибута символа в позиции курсора (запоминая предварительно старое значение атрибута и восстанавливая его при перемещении курсора). Основная программа контролирует (в цикле) состояние левой кнопки мыши и завершает свою работу при нажатии на нее, восстанавливая предварительно старый вектор прерывания.

ПРИМЕЧАНИЕ

Данный пример не предъявляет никаких особых требований к составу оборудования. Он может быть запущен на любом AT-совместимом компьютере, к которому подключена мышь типа MS Mouse.

Листинг 5.2. Поиск мыши MS Mouse и установка обработчика прерывания

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Параметры экрана в текстовом режиме
ScreenLength equ 80  ;количество символов в строке
ScreenHeigth equ 25  ;количество строк на экране

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Старое значение фона символа
```

```
OldCharBackground DB 0Fh .
; Текстовые сообщения
Txt0 DB YELLOW,0,24
     DB "Тестирование мыши типа MS Mouse",0
     DB YELLOW,24,35,"Ждите ...",0
Txt1 DB WHITE,9,26,"Мышь подключена к порту COM1",0
Txt2 DB WHITE,9,26,"Мышь подключена к порту COM2",0
Txt3 DB LIGHTGREEN,7,32,"Результаты поиска:",0
     DB LIGHTCYAN,15,26
     DB "Драйвер MS Mouse установлен",0
     DB LIGHTBLUE,17,9,"Отображение курсора "
     DB "осуществляется инверсией атрибута символа",0
     DB YELLOW,24,21
     DB "Для выхода нажмите левую клавишу мыши",0
Txt4 DB 12,26,"Мышь MS Mouse не обнаружена",0
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC MSMouseMain
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть текстовый курсор
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 2,Txt0
; Произвести поиск мыши по последовательным портам
        call    MSMouseSearch
        cmp     [COMPortNum],1  ;номер порта больше 1?
        ja      @@MouseNotFound ;мышь не найдена
; Выдать сообщение об обнаружении мыши
        cmp     [COMPortNum],0
        jne     @@COM2
        mov     SI,offset Txt1
        jmp     @@COM1
@@COM2: mov     SI,offset Txt2
@@COM1: call    ShowColorString
; Выдать сообщение о запуске драйвера
        MShowColorText 4,Txt3

; ДЕМОНСТРАЦИЯ РАБОТЫ ДРАЙВЕРА МЫШИ
; Обнулить позицию курсора мыши
```

продолжение ⇗

Листинг 5.2 (*продолжение*)

```
        mov     [XCoordinate],0
        mov     [YCoordinate],0
        mov     [OldXCoordinate],0
        mov     [OldYCoordinate],0
; Отобразить курсор мыши первый раз
        mov     [MouseCursorStatus],1
        call    ShowNewMouseCursorPosition
; Установить новый обработчик прерывания
        call    SetMSMouseInterrupt
; Цикл, пока не нажата левая кнопка
@@Next: test    [ButtonsStatus],10b
        jz      @@Next
; Восстановить прежний обработчик прерывания
        call    RestoreOldMSMouseInterrupt
; Переустановить текстовый режим и очистить экран
@@End:  mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
; Выдать сообщение "мышь не обнаружена"
@@MouseNotFound:
        MFatalError Txt4
ENDP MSMouseMain

;*****************************************
;* ОТОБРАЖЕНИЕ КУРСОРА МЫШИ ПУТЕМ ИНВЕРСИИ *
;*   АТРИБУТА СИМВОЛА В ПОЗИЦИИ КУРСОРА    *
;*****************************************
PROC ShowNewMouseCursorPosition NEAR
        pusha
        push    ES
        ; Настроить ES на видеопамять
        mov     AX,0B800h
        mov     ES,AX
        ; Вычислить старую координату курсора
        mov     AX,[OldYCoordinate]
        mov     DX,160
        mul     DX
        add     AX,[OldXCoordinate]
        add     AX,[OldXCoordinate]
        inc     AX
        mov     DI,AX
        ; Восстановить атрибут символа
        mov     AL,[OldCharBackground]
        mov     [ES:DI],AL
        ; Вычислить новую координату курсора
        mov     AX,[YCoordinate]
        mov     DX,160
        mul     DX
```

```
        add     AX,[XCoordinate]
        add     AX,[XCoordinate]
        inc     AX
        mov     DI,AX
        ; Сохранить атрибут символа
        mov     AL,[ES:DI]
        mov     [OldCharBackground],AL
        ; Инвертировать атрибут
        xor     [byte ptr ES:DI],1111111b
        ; Запомнить координаты символа
        mov     AX,[XCoordinate]
        mov     [OldXCoordinate],AX
        mov     AX,[YCoordinate]
        mov     [OldYCoordinate],AX
        pop     ES
        popa
        ret
ENDP ShowNewMouseCursorPosition
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить универсальные процедуры для работы
; с мышью MS Mouse
include "list5_01.inc"

END
```

Таинственная мышь PS/2

Мышь PS/2 — это, пожалуй, самое глубоко законспирированное периферийное устройство персонального компьютера. Протокол передачи данных этой мыши был опубликован в документации к старым моделям IBM PS/2, а вся остальная информация длительное время была недоступной. Интерес к манипуляторам с интерфейсом PS/2 возник по причине повсеместного распространения стандарта ATX, вытеснившего стандарт AT. Материнская плата формата ATX снабжена двумя разъемами PS/2-типа, один из которых предназначен для подключения клавиатуры, а второй — для координатного устройства (можно присоединять мышь, джойстик, трекбол и т. д.).

Функции BIOS для работы с мышью PS/2-типа

Для обслуживания мыши PS/2 имеется специальная функция C2h прерывания BIOS Int 15h — интерфейс координатного устройства

[3, 10, 85]. К сожалению, данный интерфейс носит вспомогательный характер — он служит только для управления параметрами мыши, а драйвер в его состав не входит.

После выполнения любой функции данной группы в случае успешного ее завершения флаг CF будет сброшен, в случае ошибки — установлен. Кроме того, в любом случае в регистр AH будет помещена дополнительная информация о результате операции:

- 00h — успешное завершение;
- 01h — недопустимый номер функции;
- 02h — недопустимое значение параметра;
- 03h — ошибка интерфейса;
- 04h — необходимо выполнить повторную передачу информации;
- 05h — драйвер устройства не установлен.

Рассмотрим набор подфункций интерфейса координатного устройства более подробно.

Прерывание Int 15h, функция C2h, подфункция 00h: разрешить или запретить работу мыши PS/2

Подфункция позволяет разрешить или запретить работу мыши или другого координатного устройства, использующего интерфейс PS/2.

Перед вызовом прерывания Int 15h необходимо занести в регистры следующие значения:

- в AX — значение C200h;
- в BH — код команды (0 — запретить работу, 1 — разрешить работу).

Прерывание Int 15h, функция C2h, подфункция 01h: произвести сброс устройства

С помощью этой подфункции осуществляется перевод мыши или другого координатного устройства в исходное состояние.

Перед вызовом прерывания необходимо занести в регистр AX значение C201h.

После сброса параметры устройства будет находиться в следующем состоянии:

- частота передачи данных: 100 пакетов в секунду;
- разрешение: 4 отсчета на миллиметр;
- масштабирование 1:1;
- работа устройства запрещена.

Прерывание Int 15h, функция C2h, подфункция 02h: установить частоту дискретизации

Подфункция позволяет изменить частоту дискретизации (частоту передачи пакетов данных от устройства к компьютеру).

Перед вызовом прерывания необходимо занести в регистры следующие значения:

- в AX — значение C202h;
- в BH — код частоты дискретизации (0 — 10 посылок/с, 1 — 20 посылок/с, 2 — 40 посылок/с, 3 — 60 посылок/с, 4 — 80 посылок/с, 5 — 100 посылок/с, 6 — 200 посылок/с).

Прерывание Int 15h, функция C2h, подфункция 03h: установить разрешение

Подфункция позволяет регулировать чувствительность мыши к перемещению.

Перед вызовом прерывания необходимо занести в регистры следующие значения:

- в AX — значение C203h;
- в BH — код устанавливаемого разрешения (0 — 1 отсчет на мм, 1 — 2 отсчета на мм, 2 — 4 отсчета на мм, 3 — 8 отсчетов на мм).

Прерывание Int 15h, функция C2h, подфункция 04h: получить идентификатор устройства

Подфункция предназначена для определения типа подключенного к компьютеру координатного устройства.

Перед вызовом прерывания необходимо занести в регистр AX значение C204h.

В случае успешного выполнения функции в регистре BH будет возвращен идентификационный код устройства.

Прерывание Int 15h, функция C2h, подфункция 05h: инициализировать интерфейс координатного устройства

Подфункция инициализирует интерфейс координатного устройства.

Перед вызовом прерывания необходимо занести в регистры следующие значения:

- в AX — значение C205h;
- в BH — размер пакета данных в байтах (от 1 до 8).

Прерывание Int 15h, функция C2h, подфункция 06h: получить состояние или установить масштаб

Подфункция позволяет определить состояние устройства или установить режим масштабирования.

Перед вызовом прерывания необходимо занести в регистры следующие значения:

- в AX — значение C206h;
- в BH — код выполняемой операции (0 — определить состояние устройства, 1 — установить масштаб 1:1, 2 — установить масштаб 2:1).

После завершения выполнения операций с кодами 1 или 2 функция возвращает только общий код состояния координатного устройства в регистре AH. В случае успешного завершения операции с кодом 0 в регистрах будут размещены следующие значения:

- в AH — общий код состояния;
- в BL — байт состояния устройства;
- в CL — код текущего разрешения устройства (0 — 1 отсчет на мм, 1 — 2 отсчета на мм, 2 — 4 отсчета на мм, 3 — 8 отсчетов на мм);
- в DL — код частоты дискретизации (0 — 10 посылок/с, 1 — 20 посылок/с, 2 — 40 посылок/с, 3 — 60 посылок/с, 4 — 80 посылок/с, 5 — 100 посылок/с, 6 — 200 посылок/с).

При этом разряды байта состояния устройства, возвращаемого в BL, имеют следующее значение:

- бит 0 — состояние правой кнопки (0 — отпущена, 1 — нажата);
- бит 1 — состояние средней кнопки (0 — отпущена, 1 — нажата);
- бит 2 — состояние левой кнопки (0 — отпущена, 1 — нажата);
- бит 3 — зарезервирован (всегда 0);
- бит 4 — масштабирование (0 — режим 1:1, 1 — режим 2:1);
- бит 5 — передача данных от устройства к компьютеру (0 — запрещена, 1 — разрешена);
- бит 6 — режим (0 — потоковый, 1 — дистанционного управления);
- бит 7 — зарезервирован (всегда 0).

Прерывание Int 15h, функция C2h, подфункция 07h: установить адрес обработчика прерываний

Функция позволяет программисту установить свою собственную процедуру обработки прерываний от координатного устройства.

Перед вызовом прерывания необходимо занести в регистры следующие значения:

- в AX — значение C207h;
- в ES:BX — сегментный адрес и смещение пользовательского обработчика прерываний от координатного устройства.

К сожалению, обработчик прерываний не входит в данный набор функций, и писать его должен сам программист — по правилам, которые мы рассмотрим далее.

Группа форматов PS/2 Mouse

Информация о работе мыши PS/2 появилась в Интернете сравнительно недавно [56, 59, 60]. Стандартный формат передачи данных для мыши PS/2-типа, разработанный фирмой IBM, показан в табл. 5.6.

Обозначения в таблице расшифровываются следующим образом:

- X0–X7 — перемещение по оси X;
- Y0–Y7 — перемещение по оси Y;
- XV — признак возникновения переполнения по X (1 — переполнение);
- YV — признак возникновения переполнения по Y (1 — переполнение);
- XS — знак перемещения по X;
- YS — знак перемещения по Y;
- M — состояние средней кнопки (0 — отпущена, 1 — нажата);
- R — состояние правой кнопки (0 — отпущена, 1 — нажата);
- L — состояние левой кнопки (0 — отпущена, 1 — нажата).

Особенность данного формата заключается в том, что координаты X и Y являются двоичными 9-разрядными числами (старший разряд — знаковый). Ось Y мыши в данном формате направлена вверх, то есть противоположно оси Y дисплея. Преимуществом формата PS2 является простота, а недостатком — отсутствие самосинхронизации

(первый байт кода не обладает отличительными признаками, позволяющими обнаружить сбой в порядке принимаемых данных, поэтому контроль приходится осуществлять другими способами).

Таблица 5.6. Стандартный формат PS/2 Mouse

Номер байта в посылке	Номер бита							
	7	6	5	4	3	2	1	0
1	YV	XV	YS	XS	1	M[1]	R	L
2	X7	X6	X5	X4	X3	X2	X1	X0
3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

[1] У стандартных двухкнопочных манипуляторов этот бит всегда равен нулю.

В течение длительного времени область применения устройств PS/2-типа была ограниченной, однако с внедрением стандарта ATX началось их интенсивное распространение и развитие. В частности, появились новые, четырехбайтные протоколы передачи данных для трехкоординатных устройств, показанные в табл. 5.7 и 5.8. При включении питания такие устройства начинают работать в стандартном режиме PS/2 со стандартным трехбайтным протоколом, а переключение в режим 3D с четырехбайтным протоколом происходит по специальной команде от драйвера мыши.

Формат 3D PS/2 Mouse предусматривает передачу координаты Z в четвертом байте пакета — в виде восьмиразрядного двоичного числа (старший разряд числа содержит знак). Формат пятикнопочной мыши Wheel Mouse отличается от предыдущего тем, что для передачи перемещения по Z применяются только четыре разряда, а еще два кодируют состояние специальных кнопок B4 и B5.

Таблица 5.7. Формат 3D PS/2 Mouse

Номер байта в посылке	Номер бита							
	7	6	5	4	3	2	1	0
1	0	0	YS	XS	1	M	R	L
2	X7	X6	X5	X4	X3	X2	X1	X0
3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
4	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0

Таблица 5.8. Формат данных 5-кнопочной мыши Wheel Mouse

Номер байта в посылке	Номер бита							
	7	6	5	4	3	2	1	0
1	0	0	YS	XS	1	M	R	L
2	X7	X6	X5	X4	X3	X2	X1	X0
3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
4	0	0	B5	B4	Z3	Z2	Z1	Z0

Непосредственная работа с мышью PS/2-типа

Мышь PS/2 подключаются не к последовательному порту, а к разъему дополнительного устройства PS/2. Мышь обслуживается тем же контроллером материнской платы, что и клавиатура, то есть получает команды и передает данные через те же порты ввода-вывода. Отличие заключается в том, что при приеме пакета данных от мыши контроллер вырабатывает прерывание IRQ12. Это прерывание необходимо закрепить за мышью с помощью процедуры BIOS SETUP, иначе оно может быть захвачено каким-либо другим устройством и станет для мыши недоступным.

Мышь PS/2 может находиться в одном из указанных ниже режимов работы.

- Потоковый режим — мышь выдает пакет данных, если произошло изменение координат устройства, нажатие или отпускание кнопки. Максимальная скорость передачи данных определяется установленной частотой дискретизации.
- Режим дистанционного управления — мышь осуществляет передачу пакета данных только по запросу со стороны компьютера, то есть по команде считывания данных Read Data.
- Эхо-режим — любой байт данных, переданный компьютером (кроме ECh и FFh), мышь отсылает обратно.

После включения напряжения питания или получения команды Reset мышь ожидает примерно 0,5 секунды и посылает компьютеру последовательность байтов: AAh, 00h. Затем по умолчанию устанавливается инкрементальный потоковый режим, масштабирование 1:1, темп передачи 100 пакетов данных в секунду. После этого мышь

отключается и больше никаких операций не производит, пока компьютер не передаст команду Enable (которая активизирует мышь).

Список команд, выполняемых мышью PS/2, приводится в табл. 5.9.

Таблица 5.9. Команды, выполняемые мышью PS/2

Код	Команда	Выполняемое действие
E6h	Reset Scaling	Отменить масштабирование перемещения
E7h	Set Scaling 2:1	Установить масштабирование перемещения для увеличения чувствительности мыши (см. табл 5 10)
E8h	Set Resolution	Установить разрешение
E9h	Status Request	Получить информацию о состоянии устройства
EAh	Set Stream Mode	Установить потоковый режим
EBh	Read Data	Выдать пакет данных
ECh	Reset Wrap Mode	Отменить эхо-режим
EDh	Set Wrap Mode	Установить эхо-режим
F0h	Set Remote Mode	Установить режим дистанционного управления
F2h	Read Device Type	Определить тип устройства
F3h	Set Sampling Rate	Установить частоту дискретизации
F4h	Enable	Разрешить передачу данных от мыши
F5h	Disable	Остановить передачу данных от мыши
F6h	Set Default	Установить значение параметров работы мыши, используемое по умолчанию
FEh	Resend	Повторить передачу данных
FFh	Reset	Осуществить сброс и произвести самотестирование устройства

Рассмотрим перечисленные команды более подробно.

- Команда E6h (Reset Scaling) отменяет режим масштабирования перемещения (устанавливает масштаб 1:1).
- Команда E7h (Set Scaling 2:1) устанавливает режим масштабирования перемещения 2:1 для увеличения чувствительности мыши (табл. 5.10).

Таблица 5.10. Масштабирование перемещения в режиме 2:1

Реальное перемещение	Информация о перемещении, выдаваемая мышью
0	0
1	1

Реальное перемещение	Информация о перемещении, выдаваемая мышью
2	1
3	3
4	6
5	9
N (≥ 6)	2N

- Команда EBh (Set Resolution) служит для установки используемого разрешения, код которого передается в следующем за командой байте данных и определяет число отсчетов на миллиметр (00h — 1 отсчет, 01h — 2 отсчета, 02h — 4 отсчета, 03h — 8 отсчетов).
- Команда E9h (Status Request) вызывает выдачу специального трехбайтного пакета данных, содержащего информацию о статусе устройства (то есть о текущих значениях параметров работы): формат первого байта показан в табл. 5.11, второй байт содержит код разрешения, третий — код частоты дискретизации.

Таблица 5.11. Формат первого байта статуса

Номер бита	Значение
0	Состояние правой кнопки (0 — отпущена, 1 — нажата)
1	Состояние средней кнопки (0 — отпущена, 1 — нажата)
2	Состояние левой кнопки (0 — отпущена, 1 — нажата)
3	Зарезервирован (всегда 0)
4	Масштабирование (0 — режим 1:1, 1 — режим 2:1)
5	Передача данных (0 — запрещена, 1 — разрешена)
6	Режим (0 — потоковый, 1 — дистанционного управления)
7	Зарезервирован (всегда 0)

- Команда EAh (Set Stream Mode) переключает мышь в потоковый режим: передача пакетов данных будет происходить при каждом изменении состояния устройства.
- Команда EBh (Read Data) вызывает (сразу по ее получении мышью) выдачу пакета данных об изменении координат и состоянии кнопок (данные передаются даже в том случае, если координаты устройства и состояние кнопок не изменялись).
- Команда ECh (Reset Wrap Mode) отключает эхо-режим, то есть служит для отмены команды EDh.

- Команда EDh (Set Wrap Mode) переводит мышь в эхо-режим. В эхо-режиме мышь отсылает обратно в компьютер каждый переданный ей байт данных (кроме ECh и FFh). Применяется данный режим для тестирования интерфейса мыши. Отменить его можно посылкой команды ECh или FFh.
- Команда F0h (Set Remote Mode) позволяет установить режим дистанционного управления: мышь будет передавать пакеты данных только по запросу со стороны компьютера (по команде EBh).
- Команда F2h (Read Device Type) предназначена для определения типа подключенного координатного устройства. Мышь должна выдавать в ответ на эту команду однобайтный код 00h.
- Команда F3h (Set Sampling Rate) служит для установки частоты дискретизации. Следующий за ней байт данных должен содержать код, задающий максимальное количество пакетов данных, которое может быть передано за одну секунду (табл. 5.12).

Таблица 5.12. Коды для установки частоты дискретизации

Код	Частота дискретизации, пакет/с
0Ah	10
14h	20
28h	40
3Ch	60
50h	80
64h	100
C8h	200

- Команда F4h (Enable) разрешает возобновить передачу данных, если мышь работает в потоковом режиме.
- Команда F5h (Disable) используется в потоковом режиме, чтобы остановить передачу данных, инициированную мышью. Если мышь работает в потоковом режиме, то передачу данных нужно запрещать перед подачей любой команды, которая предусматривает ответ (передачу данных) со стороны мыши.
- Команда F6h (Set Default) применяется, чтобы установить параметры работы по умолчанию, то есть перевести мышь в то состояние, в которое она устанавливается обычно после включения электропитания.
- Команда FEh (Resend) применяется при обнаружении любой ошибки передачи данных от мыши: получив эту команду, мышь

выполняет повторную передачу предыдущего пакета данных. Однако сигнал помехи, способный вызвать сбой при передаче данных от мыши, должен быть настолько мощным, что, скорее всего, нарушит и внутренние установки параметров встроенного в мышь микроконтроллера, поэтому при обнаружении ошибки лучше подать команду сброса и все параметры переустановить.

- Команда FFh (Reset) используется при обнаружении серьезных сбоев в работе мыши. По этой команде выполняется процедура сброса, самотестирования и выдачи идентификационной последовательности байтов (как при включении напряжения питания).

Рассмотрим примеры работы с мышью PS/2. В листинге 5.3 приведены две подпрограммы общего назначения. Процедура Wait8042BufferEmpty выполняет операцию ожидания поступления сигнала очистки входного буфера контроллера клавиатуры (ее нужно вызывать перед началом операции записи байта в буфер), а процедура WaitMouseData — операцию ожидания поступления данных от мыши.

Листинг 5.3. Процедуры общего назначения

```
CODESEG
;***********************************************
;*   ОЖИДАНИЕ ОЧИСТКИ ВХОДНОГО БУФЕРА I8042    *
;* При выходе из процедуры:                    *
;* флаг ZF установлен - нормальное завершение, *
;* флаг ZF сброшен - ошибка тайм-аута.         *
;***********************************************
proc Wait8042BufferEmpty near
        push    CX
        mov     CX,0FFFFh  ;задать число циклов ожидания
@@kb:
        in      AL,64h     ;получить статус
        test    AL,10b     ;буфер i8042 свободен?
        loopnz  @@kb       ;если нет, то цикл
        pop     CX
        ;Если при выходе из подпрограммы сброшен
        ;флаг ZF - ошибка
        ret                ;возврат в подпрограмму
endp Wait8042BufferEmpty

;**************************************
;* ОЖИДАНИЕ ПОСТУПЛЕНИЯ ДАННЫХ ОТ МЫШИ *
;**************************************
proc WaitMouseData near
        push    CX
        mov     CX,0FFFFh  ;задать число циклов ожидания
@@mouse:
        in      AL,64h     ;опросить регистр статуса
```

продолжение ⇨

Листинг 5.3 (*продолжение*)

```
        test    AL,100000b ;данные поступили?
        loopz   @@mouse    ;если нет, то цикл
        pop     CX
        ;Если при выходе из подпрограммы установлен
        ;флаг ZF - ошибка
        ret
endp WaitMouseData
ENDS
```

В листинге 5.4 показан пример программы (SetPS2MouseParameters), изменяющей характеристики работы мыши PS/2 (разрешение и частоту дискретизации). Как упоминалось выше, пакет данных, описывающий состояние мыши, состоит из трех байт: байта статуса, байта разрешения и байта частоты дискретизации. Старые и новые значения характеристик отображаются на экран в виде таблицы (в двоичном коде). Для запуска программы пригоден любой AT-совместимый персональный компьютер с подключенной мышью PS/2-типа.

ПРИМЕЧАНИЕ

Если в момент запуска программы-примера на компьютере уже установлен драйвер мыши PS/2, то после завершения работы программы драйвер сам восстановит нужные ему значения параметров.

Листинг 5.4. Управление параметрами работы мыши типа PS/2

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Текстовые сообщения
Txt1 DB YELLOW,0,18
     DB "Управление параметрами работы мыши типа PS/2",0
     DB LIGHTGREEN,10,0,"Состояние",0
```

```
    DB LIGHTGREEN,10,16,"Статус",0
    DB LIGHTGREEN,10,32,"Разрешение",0
    DB LIGHTGREEN,10,48,"Дискретизация",0
    DB LIGHTCYAN,12,0,"Исходное:",0
    DB LIGHTCYAN,14,0,"Установленное:",0
    DB YELLOW,24,22
    DB "Нажмите любую клавишу на клавиатуре",0
Err1 DB 11,22,"Ошибка обмена данными с мышью PS/2!",0
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC SetPS2MouseParameters
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText B,Txt1

; ЗАПРЕТИТЬ ГЕНЕРАЦИЮ ПРЕРЫВАНИЯ IRQ12
        ; Прочитать внутренний регистр команд i8042
        call    Wait8042BufferEmpty
        mov     AL,020h
        out     64h,AL
        in      AL,60h
        ; Сохранить содержимое регистра команд в AH
        mov     AH,AL
        ; Записать внутренний регистр команд i8042
        call    Wait8042BufferEmpty
        mov     AL,060h
        out     64h,AL
        ; Запретить сигнал прерывания от мыши
        mov     AL,AH
        and     AL,11111101b
        out     60h,AL

; ПРОЧИТАТЬ ИСХОДНОЕ СОСТОЯНИЕ МЫШИ
; Установить белый цвет символов и черный фон
        mov     [TextColorAndBackground],0Fh
; Установить начальную позицию для вывода данных на экран
        mov     [ScreenString],12
```

продолжение

Листинг 5.4 (*продолжение*)

```
        mov     [ScreenColumn],16
; Послать команду считывания состояния мыши
        call    Wait8042BufferEmpty
        mov     AL,0D4h ;послать мыши байт данных
        out     64h,AL
        call    Wait8042BufferEmpty
        mov     AL,0E9h ;команда считывания состояния
        out     60h,AL
; Получить от мыши код подтверждения приема команды
        call    WaitMouseData
        jz      @@DataInputError ;данные не поступили
        in      AL,60h
        ; Получено подтверждение приема команды?
        cmp     AL,0FAh
        jnz     @@DataInputError ;нет подтверждения приема
; Принять информацию от мыши
        ; Получить первый байт состояния
        call    WaitMouseData
        jz      @@DataInputError
        in      AL,60h
        call    ShowBinByte
        add     [ScreenColumn],8
        ; Получить второй байт состояния
        call    WaitMouseData
        jz      @@DataInputError
        in      AL,60h
        call    ShowBinByte
        add     [ScreenColumn],8
        ; Получить третий байт состояния
        call    WaitMouseData
        jz      @@DataInputError
        in      AL,60h
        call    ShowBinByte
        add     [ScreenColumn],8

; УСТАНОВИТЬ РАЗРЕШЕНИЕ
; Послать команду установки разрешения
        call    Wait8042BufferEmpty
        mov     AL,0D4h ;послать мыши байт данных
        out     64h,AL
        call    Wait8042BufferEmpty
        mov     AL,0E8h ;команда установки разрешения
        out     60h,AL
; Получить от мыши код подтверждения приема команды
        call    WaitMouseData
        jz      @@DataInputError ;данные не поступили
        in      AL,60h
        ; Получено подтверждение приема команды?
        cmp     AL,0FAh
```

```
        jnz     @@DataInputError ;нет подтверждения
; Послать код разрешения
        call    Wait8042BufferEmpty
        mov     AL,0D4h
        out     64h,AL
        call    Wait8042BufferEmpty
        mov     AL,1    ;код разрешения 1 точка/мм
        out     60h,AL
; Получить код подтверждения приема команды
        call    WaitMouseData
        jz      @@DataInputError ;данные не поступили
        in      AL,60h
        ; Получено подтверждение приема команды?
        cmp     AL,0FAh
        jnz     @@DataInputError ;нет подтверждения

; УСТАНОВИТЬ ЧАСТОТУ ДИСКРЕТИЗАЦИИ
; Послать команду установки частоты дискретизации
        call    Wait8042BufferEmpty
        mov     AL,0D4h ;послать мыши байт данных
        out     64h,AL
        call    Wait8042BufferEmpty
        mov     AL,0F3h ;установить частоту
        out     60h,AL
; Получить код подтверждения приема команды
        call    WaitMouseData
        jz      @@DataInputError ;данные не поступили
        in      AL,60h
        ; Получено подтверждение приема команды?
        cmp     AL,0FAh
        jnz     @@DataInputError ;нет подтверждения
; Послать код частоты дискретизации
        call    WaitB042BufferEmpty
        mov     AL,0D4h
        out     64h,AL
        call    Wait8042BufferEmpty
        mov     AL,0C8h ;выдавать 200 пакетов в секунду
        out     60h,AL
; Получить код подтверждения приема команды
        call    WaitMouseData
        jz      @@DataInputError ;данные не поступили
        in      AL,60h
        ; Получено подтверждение приема команды?
        cmp     AL,0FAh
        jnz     @@DataInputError ;нет подтверждения

; ПРОЧИТАТЬ НОВОЕ СОСТОЯНИЕ МЫШИ
        mov     [ScreenString],14
        mov     [ScreenColumn],16
; Послать команду считывания состояния мыши
```

продолжение

Листинг 5.4 (*продолжение*)

```
        call    Wait8042BufferEmpty
        mov     AL,0D4h
        out     64h,AL
        call    Wait8042BufferEmpty
        mov     AL,0E9h
        out     60h,AL
; Получить код подтверждения приема команды
        call    WaitMouseData
        jz      @@DataInputError ;данные не поступили
        in      AL,60h
        ; Получено подтверждение приема команды?
        cmp     AL,0FAh
        jnz     @@DataInputError ;нет подтверждения
; Принять информацию от мыши
        ; Получить первый байт состояния
        call    WaitMouseData
        jz      @@DataInputError
        in      AL,60h
        call    ShowBinByte
        add     [ScreenColumn],8
        ; Получить второй байт состояния
        call    WaitMouseData
        jz      @@DataInputError
        in      AL,60h
        call    ShowBinByte
        add     [ScreenColumn],8
        ; Получить третий байт состояния
        call    WaitMouseData
        jz      @@DataInputError
        in      AL,60h
        call    ShowBinByte
        add     [ScreenColumn],8

; РАЗРЕШИТЬ ГЕНЕРАЦИЮ ПРЕРЫВАНИЯ IRQ12
        ; Прочитать внутренний регистр команд i8042
        call    Wait8042BufferEmpty
        mov     AL,020h
        out     64h,AL
        in      AL,60h
        ; Записать внутренний регистр команд i8042
        mov     AH,AL
        call    Wait8042BufferEmpty
        mov     AL,060h
        out     64h,AL
        mov     AL,AH
        or      AL,10b ;разрешить прерывание от мыши
        out     60h,AL
```

```
        jmp short @@End

; Ожидать нажатия любой клавиши на клавиатуре
@@End:  call    GetChar
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h

; Обработка ошибок
@@DataInputError:
        MFatalError Err1
ENDP SetPS2MouseParameters
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуры очистки выходного буфера
; контроллера клавиатуры и ожидания поступления
; данных от мыши
include "list5_03.inc"

END
```

В листинге 5.5 приведена программа PS2MouseStart, устанавливающая новый (пользовательский) обработчик прерывания от мыши PS/2. Прежде чем установить обработчик при помощи процедуры SetPS2MouseInterrupt, основная программа посылает мыши команду разрешения передачи данных, одновременно проверяя тем самым наличие мыши и ее исправность (если мышь не отвечает кодом подтверждения, значит, она не подключена).

Программа обработки прерывания PS2MouseInterrupt принимает пакеты данных от мыши и после приема каждого трехбайтного пакета определяет состояние кнопок и приращение значений координат курсора. Для вывода курсора на экран используется процедура ShowNewMouseCursorPosition, которая инвертирует байт атрибута символа в позиции курсора (запоминая предварительно старое значение атрибута и восстанавливая его при перемещении курсора). Основная программа контролирует (в цикле) состояние левой кнопки мыши и завершает свою работу при нажатии на нее, восстанавливая предварительно старый вектор прерывания при помощи процедуры RestoreOldPS2MouseInterrupt. Для запуска программы пригоден любой AT-совместимый персональный компьютер с подключенной мышью PS/2-типа.

Листинг 5.5. Установка нового обработчика прерывания мыши PS/2, отображающего указатель в текстовом режиме инверсией атрибута символа

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

ScreenLength equ 80
ScreenWidth  equ 25

DATASEG
; Область сохранения старого вектора прерывания мыши
OldPS2MouseInterruptOffset  DW ?
OldPS2MouseInterruptSegment DW ?
; Номер принимаемого от мыши байта
MouseByteNumber DB 0
; Трехбайтовая структура данных, передаваемая мышью
FirstByte       DB 0
SecondByte      DB 0
ThirdByte       DB 0
; Текущее состояние кнопок
ButtonsStatus   DB 0
; Текущие координаты курсора мыши
XCoordinate     DW 0
YCoordinate     DW 0
; Предыдущая позиция курсора мыши
OldXCoordinate  DW 0
OldYCoordinate  DW 0
; Старое значение фона символа
OldCharBackground DB 0Fh
; Текстовые сообщения
Txt1 DB LIGHTGREEN,0,28,"Тест для мыши PS/2-типа",0
     DB WHITE,12,14,"Отображение курсора "
     DB "мыши инверсией атрибута символа",0
     DB YELLOW,24,21
     DB "Для выхода нажмите левую клавишу мыши",0
Err1 DB 11,22,"Ошибка обмена данными с мышью PS/2!",0
ENDS
```

```
CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC PS2MouseStart
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть текстовый курсор - убрать за
; нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 3,Txt1

; ЗАПРЕТИТЬ ГЕНЕРАЦИЮ ПРЕРЫВАНИЯ IRQ12
        ; Прочитать внутренний регистр команд i8042
        call    Wait8042BufferEmpty
        mov     AL,020h
        out     64h,AL
        in      AL,60h
        ; Сохранить содержимое регистра команд в AH
        mov     AH,AL
        ; Записать внутренний регистр команд i8042
        call    Wait8042BufferEmpty
        mov     AL,060h
        out     64h,AL
        ; Запретить сигнал прерывания от мыши
        mov     AL,AH
        and     AL,11111101b
        out     60h,AL

; РАЗРЕШИТЬ ПЕРЕДАЧУ ДАННЫХ ОТ МЫШИ
        call    Wait8042BufferEmpty
        mov     AL,0D4h ;послать мыши байт данных
        out     64h,AL
        call    Wait8042BufferEmpty
        mov     AL,0F4h ;разрешить передачу данных
        out     60h,AL
; Получить код подтверждения приема команды
        call    WaitMouseData
        jz      @@DataInputError ;данные не поступили
        in      AL,60h
        ; Получено подтверждение приема команды?
        cmp     AL,0FAh
        jnz     @@DataInputError ;нет подтверждения
```

продолжение ⇒

Листинг 5.5 (*продолжение*)

```
; РАЗРЕШИТЬ ГЕНЕРАЦИЮ ПРЕРЫВАНИЯ IRQ12
; Прочитать внутренний регистр команд i8042
        call    Wait8042BufferEmpty
        mov     AL,020h
        out     64h,AL
        in      AL,60h
; Записать внутренний регистр команд i8042
        mov     AH,AL
        call    Wait8042BufferEmpty
        mov     AL,060h
        out     64h,AL
; Разрешить сигнал прерывания от мыши
        mov     AL,AH
        or      AL,10b
        out     60h,AL

; УСТАНОВИТЬ НОВЫЙ ОБРАБОТЧИК ПРЕРЫВАНИЯ
; Обнулить позицию курсора мыши
        mov     [XCoordinate],0
        mov     [YCoordinate],0
        mov     [OldXCoordinate],0
        mov     [OldYCoordinate],0
; Отобразить курсор мыши первый раз
        call    ShowNewMouseCursorPosition
; Установить новый обработчик прерывания
        call    SetPS2MouseInterrupt
; Цикл, пока не нажата левая кнопка
@@Next: test    [ButtonsStatus],1b
        jz      @@Next
        jmp short @@End

; ЗАВЕРШЕНИЕ РАБОТЫ ПРОГРАММЫ
; Восстановить прежний обработчик прерывания
@@End:  call    RestoreOldPS2MouseInterrupt
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h

; Обработка ошибок
@@DataInputError:
        MFatalError Err1
ENDP PS2MouseStart

;***********************************
;* ОБРАБОТЧИК ПРЕРЫВАНИЯ ОТ МЫШИ PS/2 *
;***********************************
```

```
proc PS2MouseInterrupt far
        sti     ;разрешить маскируемые прерывания
        pusha
        push    DS
        mov     AX,[CS:MainDataSeg]
        mov     DS,AX
        call    Wait8042BufferEmpty ;очистка буфера
        in      AL,60h              ;получить скан-код

; Выбирать порядковый номер принимаемого байта
        cmp     [MouseByteNumber],0
        je      @@SaveFirstByte
        cmp     [MouseByteNumber],1
        je      @@SaveSecondByte
        cmp     [MouseByteNumber],2
        je      @@SaveThirdByte
        jmp     @@Error
; Записать первый байт посылки
@@SaveFirstByte:
        test    AL,1000b    ;первый байт посылки?
        jz      @@Error     ;сбой синхронизации
        mov     [FirstByte],AL
        inc     [MouseByteNumber]
        jmp     @@EndMouseInterrupt
; Записать второй байт посылки
@@SaveSecondByte:
        mov     [SecondByte],AL
        inc     [MouseByteNumber]
        jmp     @@EndMouseInterrupt
; Записать третий байт посылки
@@SaveThirdByte:
        mov     [ThirdByte],AL
        mov     [MouseByteNumber],0
; (пакет данных от мыши принят полностью)

; Записать новое значение байта состояния кнопок
        mov     AL,[FirstByte]
        and     AL,0111b
        mov     [ButtonsStatus],AL
; Вычислить новую X-координату курсора
        ; Занести в AX перемещение по X
        mov     AH,0   ;дублируем знак во все разряды AH
        mov     AL,[FirstByte]
        test    AL,10000b
        jz      @@M0
        mov     AH,0FFh
        ; Занести в AL младший байт
@@M0:   mov     AL,[SecondByte]
        ; Вычислить новое значение координаты
        ; курсора по X
```

продолжение ⇩

Листинг 5.5 (*продолжение*)

```
        add     AX,[XCoordinate]
        cmp     AX,0
        jge     @@M1
        mov     AX,0
        jmp short @@M2
@@M1:   cmp     AX,ScreenLength
        jl      @@M2
        mov     AX,ScreenLength-1
@@M2:   mov     [XCoordinate],AX

; Вычисляем новую Y-координату курсора
        ; Занести в AX перемещение по Y
        mov     AH,0   ;дублируем знак во все разряды AH
        mov     AL,[FirstByte]
        test    AL,100000b
        jz      @@M3
        mov     AH,0FFh
        ; Занести в AL младший байт
@@M3:   mov     AL,[ThirdByte]
        ; Вычислить новое значение координаты курсора
        ; по Y (Y-координата мыши PS/2 направлена
        ; противоположно экранной)
        neg     AX
        add     AX,[YCoordinate]
        cmp     AX,0
        jge     @@M4
        mov     AX,0
        jmp short @@M5
@@M4:   cmp     AX,ScreenWidth
        jl      @@M5
        mov     AX,ScreenWidth-1
@@M5:   mov     [YCoordinate],AX

; Показать курсор в новой позиции
        call    ShowNewMouseCursorPosition
        jmp short @@EndMouseInterrupt

; Обнаружен сбой в порядке передачи информации от мыши
@@Error:
        mov   [MouseByteNumber],0
; Нормальное завершение прерывания
@@EndMouseInterrupt:
        cli
        ; разрешение прерываний с меньшими приоритетами
        mov     AL,20h
        out     0A0h,AL ;команда EOI в i8059-2
        out     20h,AL  ;команда EOI в i8059-1
        pop     DS
```

```
        popa
        iret
endp PS2MouseInterrupt

;*****************************************
;* УСТАНОВИТЬ НОВЫЙ ВЕКТОР ПРЕРЫВАНИЯ МЫШИ *
;*****************************************
PROC SetPS2MouseInterrupt NEAR
        pusha
        push    ES
        mov     AX,0
        mov     ES,AX
; Запомнить прежний вектор обработчика
; прерывания мыши
        mov     AX,[ES:74h*4]
        mov     [OldPS2MouseInterruptOffset],AX
        mov     AX,[ES:74h*4+2]
        mov     [OldPS2MouseInterruptSegment],AX
; Установка нового вектора прерывания
        cli     ;запретить прерывания
        mov     AX,offset PS2MouseInterrupt
        mov     [ES:74h*4],AX
        mov     AX,CS
        mov     [ES:74h*4+2],AX
        sti     ;разрешить прерывания
; Размаскировать прерывание IRQ12
        in      AL,0A1h
        and     AL,11101111b
        jmp short $+2       ;задержка
        jmp short $+2       ;задержка
        out     0A1h,AL
        pop     ES
        popa
        ret
ENDP SetPS2MouseInterrupt

;*********************************************
;* ВОССТАНОВИТЬ СТАРЫЙ ВЕКТОР ПРЕРЫВАНИЯ МЫШИ *
;*********************************************
PROC RestoreOldPS2MouseInterrupt NEAR
        pusha
        push    ES
        mov     AX,0
        mov     ES,AX
; Восстановить прежний вектор обработчика прерывания
        cli
        mov     AX,[OldPS2MouseInterruptOffset]
        mov     [ES:74h*4],AX
        mov     AX,[OldPS2MouseInterruptSegment]
        mov     [ES:74h*4+2],AX
```

продолжение ⇩

Листинг 5.5 (*продолжение*)

```
        sti
        pop     ES
        popa
        ret
ENDP RestoreOldPS2MouseInterrupt

;******************************************
;* ОТОБРАЖЕНИЕ КУРСОРА МЫШИ ПУТЕМ ИНВЕРСИИ *
;*   АТРИБУТА СИМВОЛА В ПОЗИЦИИ КУРСОРА    *
;******************************************
PROC ShowNewMouseCursorPosition NEAR
        pusha
        push    ES
; Настроить ES на текстовый буфер
        mov     AX,0B800h
        mov     ES,AX
; Вычислить адрес предыдущей позиции курсора мыши
        mov     AX,[OldYCoordinate]
        mov     DX,160
        mul     DX
        add     AX,[OldXCoordinate]
        add     AX,[OldXCoordinate]
        inc     AX
        ; Занести смещение в индексный регистр
        mov     DI,AX
        ; Восстановить исходный атрибут символа
        mov     AL,[OldCharBackground]
        mov     [ES:DI],AL
; Вычислить адрес новой позиции курсора мыши
        mov     AX,[YCoordinate]
        mov     DX,160
        mul     DX
        add     AX,[XCoordinate]
        add     AX,[XCoordinate]
        inc     AX
        ; Занести смещение в индексный регистр
        mov     DI,AX
        ; Сохранить значение атрибута символа
        mov     AL,[ES:DI]
        mov     [OldCharBackground],AL
        ; Инвертировать атрибут символа (кроме
        ; старшего разряда)
        xor     [byte ptr ES:DI],1111111b
        ; Запомнить координаты курсора
        mov     AX,[XCoordinate]
        mov     [OldXCoordinate],AX
        mov     AX,[YCoordinate]
        mov     [OldYCoordinate],AX
```

```
        pop     ES
        popa
ENDP ShowNewMouseCursorPosition
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуры очистки выходного буфера
; контроллера клавиатуры и ожидания поступления
; данных от мыши
include "list5_03.inc"

END
```

ПРИМЕЧАНИЕ

Для успешного запуска примеров из листингов 5.4 и 5.5 необходимо, чтобы к компьютеру была подключена мышь PS/2-типа. Кроме того, в BIOS SETUP должно быть разрешено использование мыши PS/2 и за мышью должно быть закреплено прерывание IRQ12.

Глава 6
Работа с дисками

Для долговременного хранения информации на персональных компьютерах обычно применяются дисководы различных типов: запоминающие устройства, построенные на других конструктивных принципах, либо не обеспечивают достаточной скорости доступа к информации, либо являются слишком дорогими для массового применения.

Ниже рассматриваются способы работы с дисководами как на уровне функций операционной системы, так и на уровне аппаратного обеспечения: функции MS-DOS обеспечивают достаточную для большинства прикладных задач скорость доступа к данным, но для получения максимальной производительности приходится работать непосредственно с контроллерами дисководов.

Группа дисковых функций MS-DOS

В эту группу входят прерывания, предназначенные для выполнения основных функций операционной системы, в том числе для выполнения операций с логическими дисками, файлами и каталогами [3, 10]. Дисковые функции DOS обладают достаточной полнотой и универсальностью для решения любых задач в реальном режиме DOS. Они могут применяться и в режиме линейной адресации памяти, но информацию в расширенную память приходится пересылать через промежуточный буфер в первом мегабайте адресного пространства процессора. Впрочем, дополнительные пересылки не особенно замедляют работу: поиск данных на диске и передача информации между диском и процессором занимает гораздо больше времени, чем копирование такого же объема данных с одного участка оперативной памяти в другой.

Ниже описаны функции DOS, выполняющие основные операции над логическими дисками, каталогами и файлами. При описании используются следующие термины:

- строка ASCIIZ — текстовая строка в ASCII-коде, которая завершается нулевым значением;
- дескриптор файла — уникальный номер, который операционная система присваивает создаваемому или открываемому файлу в качестве идентификатора (чтобы потом обращаться к файлу по этому номеру вплоть до его закрытия).

Классические функции для работы с дисками

К этой группе относятся функции, появившиеся в ранних версиях операционной системы MS-DOS и сохранившиеся с тех пор практически без изменений. Такие функции отличаются крайне примитивной обработкой ошибок:

- в случае успешного завершения операции флаг CF сбрасывается в 0;
- в случае ошибки флаг CF устанавливается в 1.

Для обращения к дисковым функциям DOS используется прерывание Int 21h.

Прерывание Int 21h, функция 0Eh: сменить текущий логический диск

Функция позволяет выбрать логический диск.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 0Eh;
- в AL — код логического диска (0 — A:, 1 — B: и т. д.).

После завершения операции функция возвращает в регистре AL максимально возможный в данной системе номер логического дисковода (определяется параметром LASTDRIVE в файле CONFIG.SYS).

Прерывание Int 21h, функция 19h: определить номер текущего дисковода

Функция определяет номер дисковода, который в данный момент считается текущим, то есть используется по умолчанию.

Перед вызовом прерывания требуется записать в регистр AH значение 19h.

После завершения операции функция возвращает в регистре AL код логического диска (0 — A:, 1 — B: и т. д.).

Прерывание Int 21h, функция 1Ah: изменить адрес области обмена с диском

Функция устанавливает адрес буфера, используемого в операциях ввода-вывода и поиска в каталогах.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 1Ah;
- в DS:DX — указатель на новый адрес буфера обмена DTA.

ПРИМЕЧАНИЕ

При запуске программы ее область DTA первоначально установлена по адресу PSP:0080h.

Прерывание Int 21h, функция 2Fh: получить адрес области обмена с диском

Функция определяет текущий адрес буфера, используемого в операциях ввода-вывода и поиска в каталогах.

Перед вызовом прерывания требуется записать в регистр AH значение 2Fh.

После завершения операции функция возвращает в ES:BX указатель на адрес буфера обмена DTA.

Прерывание Int 21h, функция 36h: определить объем свободного места на диске

Функция определяет объем свободного места на заданном логическом диске.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 36h;
- в AL — код логического диска (0 — A:, 1 — B: и т. д.).

В случае ошибки в регистре AX будет возвращен код 0FFFFh (недопустимый код логического диска).

В случае успешного завершения операции функция возвращает:

- в AX — число секторов в кластере;
- в BX — число свободных кластеров;
- в CX — число байтов в секторе;
- в DX — полное число кластеров на диске.

Объем свободного пространства определяется произведением содержимого регистров AX, BX и CX, а полный объем диска в байтах — произведением AX, CX и DX.

Улучшенные функции для работы с дисками

По мере развития MS-DOS в набор функций постоянно вносились дополнения, упрощающие выполнение тех или иных операций и улучшающие контроль за их выполнением. Для вызова функций данной группы также используется прерывание Int 21h.

Перечисленные ниже функции DOS имеют усовершенствованные средства контроля: в случае ошибки, кроме установки флага CF, выдают в регистре AX код ошибки, по которому можно определить причину ее возникновения. Возможные значения кодов ошибок приведены в табл. 6.1. Однако следует учитывать, что содержимое регистра AX в случае успешного завершения данных функций не сохраняется.

Таблица 6.1. Значения расширенных кодов ошибки

Код ошибки	Расшифровка кода
00h	Нет ошибки
01h	Неверный номер функции
02h	Файл не найден
03h	Путь не найден
04h	Открыто слишком много файлов (нет свободных дескрипторов)
05h	Доступ запрещен
06h	Недопустимый дескриптор
07h	Разрушен блок управления памятью
08h	Недостаточно памяти
09h	Недопустимый адрес блока памяти
0Ah	Ошибка окружения (длина больше 32 Кбайт)
0Bh	Недопустимый формат
0Ch	Недопустимый код доступа

продолжение ⇗

Таблица 6.1 (*продолжение*)

Код ошибки	Расшифровка кода
0Dh	Недопустимые данные
0Eh	Неизвестное устройство
0Fh	Недопустимый дисковод
10h	Попытка удалить текущий каталог
11h	Устройство не то же самое
12h	Больше нет файлов
13h	Диск защищен от записи
14h	Неизвестное устройство
15h	Дисковод не готов
16h	Неизвестная команда
17h	При проверке CRC обнаружена ошибка данных
18h	Неверная длина структуры запроса
19h	Ошибка поиска (позиционирования)
1Ah	Неизвестный тип носителя (не DOS-диск)
1Bh	Сектор не найден
1Ch	В принтере нет бумаги
1Dh	Ошибка записи
1Eh	Ошибка чтения
1Fh	Общий сбой
20h	Нарушение разделения
21h	Нарушение запирания
22h	Недопустимая смена диска (в ES:DI будет находиться указатель на метку нужного тома в виде строки ASCIIZ)
23h	FCB недоступен
24h	Разделяемый буфер переполнен
25h	Несоответствие кодовой страницы
26h	Невозможно завершить операцию с файлом
27h	Недостаточно места на диске
28h–31h	Зарезервированы
32h	Неподдерживаемый сетевой запрос
33h	Удаленный компьютер не слушает
34h	Дублирование имени в сети
35h	Не найдено сетевое имя
36h	Сеть занята
37h	Сетевое устройство больше не существует
38h	Исчерпан лимит команд NetBIOS
39h	Аппаратная ошибка сетевого адаптера

Код ошибки	Расшифровка кода
3Ah	Неверный отклик из сети
3Bh	Неожиданная ошибка сети
3Ch	Несовместимый удаленный адаптер
3Dh	Очередь на печать заполнена
3Eh	Нет места для файла печати
3Fh	Файл печати удален
40h	Сетевое имя удалено
41h	Доступ к сети невозможен
42h	Неверный тип сетевого устройства
43h	Сетевое имя не найдено
44h	Исчерпан лимит сетевых имен
45h	Исчерпан лимит сеанса работы NetBIOS
46h	Временная пауза
47h	Сетевой запрос не принят
48h	Приостановлено переназначение принтера или диска
49h–4Fh	Зарезервированы
50h	Файл уже существует
51h	Зарезервирован
52h	Каталог не может быть создан
53h	Отказ по прерыванию Int 24h (критическая ошибка)
54h	Слишком много переназначений
55h	Двойное перенаправление
56h	Недопустимый пароль
57h	Недопустимый параметр
58h	Ошибка сетевой записи
59h	Функция не поддерживается в сети
5Ah	Требуемый компонент системы не установлен

Прерывание Int 21h, функция 39h: создать подкаталог

Функция создает в текущем дереве каталогов новый подкаталог.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 39h;
- в DS DX — указатель на полную спецификацию каталога в виде строки ASCIIZ (должны существовать все каталоги на заданном

пути, кроме последнего; вызов функции завершается ошибкой, если родительский каталог заполнен и является корневым).

Возможные коды ошибки: 03h, 05h.

Прерывание Int 21h, функция 3Ah: удалить подкаталог

Функция удаляет указанный подкаталог.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 3Ah;
- в DS:DX — указатель на полную спецификацию удаляемого каталога в виде строки ASCIIZ.

Возможные коды ошибки: 03h, 05h, 06h, 10h.

ПРИМЕЧАНИЕ

Каталог должен быть пустым, иначе выдается сообщение об ошибке. Поэтому перед удалением каталога нужно его очистить, удалив все имеющиеся в нем файлы и каталоги.

Прерывание Int 21h, функция 3Bh: перейти в другой каталог

Функция создает в текущем дереве каталогов новый подкаталог.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 3Bh;
- в DS:DX — указатель на полную спецификацию заданного каталога в виде строки ASCIIZ.

Возможный код ошибки: 03h.

Прерывание Int 21h, функция 3Ch: создать файл

Функция создает файл для записи. Если файл уже существует, то его размер усекается до 0.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 3Ch;
- в CX — атрибуты создаваемого файла (табл. 6.2);
- в DS:DX — указатель на имя файла в виде строки ASCIIZ.

В случае успешного завершения операции функция возвращает в регистре AX дескриптор файла.

Возможные коды ошибки: 03h, 04h, 05h.

Таблица 6.2. Формат слова атрибутов файла

Номер разряда	Описание
0	Только для чтения
1	Скрытый
2	Системный
3	Метка тома (разряд может быть установлен только при считывании атрибутов; при создании файла или изменении атрибутов значение разряда должно быть равно 0)
4	Каталог (разряд может быть установлен только при считывании атрибутов; при создании файла или изменении атрибутов значение разряда должно быть равно 0)
5	Признак архивации
6–15	Зарезервированы, должны быть равны 0

Прерывание Int 21h, функция 3Dh: открыть существующий файл

Функция открывает файл для чтения, записи или дозаписи информации. Указатель при этом устанавливается в начало файла.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 3Dh;
- в AL — режимы доступа (табл. 6.3);
- в DS:DX — указатель на имя файла в виде строки ASCIIZ.

В случае успешного завершения операции функция возвращает в регистре AX дескриптор файла.

Возможные коды ошибки: 01h, 02h, 03h, 04h, 05h, 0Ch.

Таблица 6.3. Формат байта режимов доступа

Номер разряда	Описание
0–2	Режим доступа: 000 — только для чтения; 001 — только для записи;

продолжение ⇗

Таблица 6.3 (*продолжение*)

Номер разряда	Описание
	010 — для чтения и записи
3	Зарезервирован, должен быть равен 0
4–6	Режим разделения файлов:
	000 — режим совместимости;
	001 — другим программам запрещен любой доступ к файлу;
	010 — другим программам запрещена запись в файл;
	011 — другим программам запрещено чтение из файла;
	001 — другим программам разрешен полный доступ к файлу
7	Флаг наследования:
	0 — дочерний процесс наследует дескриптор,
	1 — не наследует

Прерывание Int 21h, функция 3Eh: закрыть файл

Функция сбрасывает на диск содержимое всех буферов, обновляет информацию в каталоге, а затем освобождает дескриптор файла (после этого дескриптор может быть присвоен другому файлу).

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 3Eh;
- в BX — дескриптор файла.

Возможный код ошибки: 06h.

Прерывание Int 21h, функция 3Fh: чтение информации из файла

Функция считывает данные из файла в указанный буфер. Считывание начинается с текущей позиции указателя файла.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 3Fh;
- в BX — дескриптор файла;
- в CX — число байтов, подлежащих считыванию;
- в DS DX — указатель на буфер, в который должна быть занесена считанная информация.

В случае успешного завершения операции функция возвращает в регистре AX число реально считанных байтов (оно может быть меньше значения, указанного в регистре CX при вызове функции, если в процессе считывания достигнут конец файла).

Возможные коды ошибки: 05h, 06h.

Прерывание Int 21h, функция 40h: запись информации в файл

Функция записывает данные из указанного буфера в файл. Запись начинается с текущей позиции указателя файла.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 40h;
- в BX — дескриптор файла;
- в CX — число байтов, подлежащих записи;
- в DS:DX — указатель на буфер, информация из которого записывается в файл.

В случае успешного завершения операции функция возвращает в регистре AX число реально записанных байтов (оно может быть меньше значения, указанного в регистре CX при вызове функции, если на диске недостаточно свободного места, то есть произошло переполнение).

Возможные коды ошибки: 05h, 06h.

Прерывание Int 21h, функция 41h: удалить файл

Удаляет указанный файл.

ПРИМЕЧАНИЕ

Функция не допускает групповых операций и не удаляет файлы, имеющие атрибут «только для чтения» (этот атрибут нужно изменить при помощи функции 43h, чтобы файл можно было удалить).

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 41h;
- в DS:DX — указатель на имя файла в виде строки ASCIIZ.

Возможные коды ошибки: 02h, 03h, 05h.

Прерывание Int 21h, функция 42h: изменить положение указателя файла

Функция смещает указатель на заданное число байтов.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 42h;
- в AL — код точки отсчета положения указателя (0 — отсчет ведется от начала файла, 1 — от текущего положения указателя, 2 — от конца файла);
- в BX — дескриптор файла;
- в CX — старшая часть смещения;
- в DX — младшая часть смещения.

В случае успешного завершения операции функция возвращает в регистрах DX и AX новое положение указателя относительно начала файла:

- в DX — старшая часть значения положения указателя;
- в AX — младшая часть значения.

Возможные коды ошибки: 01h, 06h.

Прерывание Int 21h, функция 43h, подфункция 00h: получить атрибуты файла

Подфункция определяет атрибуты указанного файла.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AX — значение 4300h;
- в DS:DX — указатель на имя файла в виде строки ASCIIZ.

В случае успешного завершения операции функция возвращает в регистре CX слово атрибутов файла (см. табл. 6.2).

Возможные коды ошибки: 01h, 02h, 03h, 05h.

Прерывание Int 21h, функция 43h, подфункция 01h: изменить атрибуты файла

Подфункция изменяет атрибуты указанного файла.

ПРИМЕЧАНИЕ

Функция не может изменить атрибуты метки тома или каталога.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AX — значение 4301h;
- в CX — новые значения атрибутов файла (см. табл. 6.2);
- в DS:DX — указатель на имя файла в виде строки ASCIIZ.

Возможные коды ошибки: 01h, 02h, 03h, 05h.

Прерывание Int 21h, функция 47h: определить имя текущего каталога на указанном устройстве

Выдает имя текущего (рабочего) каталога на указанном логическом диске.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 47h;
- в DL — код логического диска (0 — диск, используемый по умолчанию (текущий), 1 — диск A:, 2 — диск B: и т. д.);
- в DS:SI — указатель на буфер размером 64 байта, выделенный для записи имени каталога.

В случае успешного выполнения функции в буфер будет записан путь от корневого каталога до текущего в виде строки ASCIIZ. Описание пути не включает в себя идентификатор диска и начальный обратный слэш «\».

Возможный код ошибки: 0Fh.

Прерывание Int 21h, функция 4Eh: найти первый файл заданного типа

Функция ищет в указанном каталоге первый файл, соответствующий заданной спецификации.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 4Eh;
- в CX — маску атрибутов файла (см. табл. 6.2);
- в DS:DX — указатель на спецификацию файла в виде строки ASCIIZ (спецификация может включать в себя путь и шаблоны).

В случае успешного завершения функция возвращает в области DTA блок данных для первого найденного файла. Формат блока данных приведен в табл. 6.4.

Возможные коды ошибки: 02h, 03h, 12h.

ПРИМЕЧАНИЕ

Получить адрес области обмена с диском DTA можно при помощи функции 2Fh.

Таблица 6.4. Формат блока данных файла для функций поиска

Смещение	Размер элемента	Описание
00h	BYTE	Буква логического диска
01h	11 байт	Шаблон для поиска
0Ch	BYTE	Атрибуты поиска
0Dh	WORD	Счетчик элементов внутри каталога
0Fh	WORD	Номер кластера начала родительского каталога
11h	4 байта	Зарезервировано
15h	BYTE	Атрибуты найденного файла
16h	WORD	Время создания файла: биты 0–4 — двухсекундные приращения: биты 5–10 — минуты; биты 11–15 — часы
18h	WORD	Дата создания файла: биты 0–4 — день; биты 5–8 — месяц; биты 9–15 — номер года (относительно 1980 года)
1Ah	DWORD	Размер файла в байтах
1Eh	13 байтов	Имя и расширение файла в виде строки ASCIIZ

Прерывание Int 21h, функция 4Fh: найти следующий файл

Функция выполняет поиск следующего файла, соответствующего спецификации, заданной при предшествующем вызове функции 4Eh.

Перед вызовом прерывания требуется записать в регистр AH значение 4Fh. В области DTA должен находиться блок данных от предыдущего вызова функции 4Eh или 4Fh.

В случае успешного завершения функция возвращает в области DTA блок данных очередного найденного файла (см. табл. 6.4).

Возможный код ошибки: 12h.

Прерывание Int 21h, функция 56h: переименовать или переместить файл

Функция изменяет текущее имя файла на заданное и может выполнить при этом перемещение файла из одного каталога в другой (в пределах одного логического диска). Возможно также переименование (но не перемещение) каталогов.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 56h;
- в DS:DX — указатель на спецификацию существующего файла в виде строки ASCIIZ (спецификация может включать в себя путь, но не должна включать шаблоны групповых операций);
- в ES:BX — новое имя файла в виде строки ASCIIZ (может включать в себя путь, но не должна включать шаблоны).

Возможные коды ошибки: 02h, 03h, 05h, 11h.

Прерывание Int 21h, функция 57h, подфункция 00h: получить время и дату создания файла

Считывает время и дату создания файла с заданным дескриптором.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AX — значение 5700h;
- в BX — дескриптор файла.

В случае успешного завершения операции функция возвращает в регистрах следующие значения:

- в CX — время создания файла:
  - биты 0–4 — двухсекундные приращения;
  - биты 5–10 — минуты;
  - биты 11–15 — часы;
- в DX — дату создания файла:
  - биты 0–4 — день;
  - биты 5–8 — месяц;
  - биты 9–15 — номер года (относительно 1980 года).

Возможные коды ошибки: 01h, 06h.

Прерывание Int 21h, функция 57h, подфункция 01h: изменить время и дату создания файла

Изменяет время и дату создания файла с заданным дескриптором.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AX — значение 5701h;
- в BX — дескриптор файла;
- в CX — время создания файла:
  - биты 0–4 — двухсекундные приращения;
  - биты 5–10 — минуты;
  - биты 11–15 — часы;
- в DX — дату создания файла:
  - биты 0–4 — день;
  - биты 5–8 — месяц;
  - биты 9–15 — номер года (относительно 1980 года).

Возможные коды ошибки: 01h, 06h.

Прерывание Int 21h, функция 59h: получить дополнительную информацию об ошибке

Эту функцию (при необходимости) следует вызывать сразу после получения сообщения о возникновении ошибки при выполнении какой-либо операции по прерыванию Int 21h. Функция выдает дополнительную информацию о причинах возникновения ошибки и рекомендации по ее устранению.

ПРИМЕЧАНИЕ

При выдаче кода ошибки обычно он упрощается до старого набора кодов DOS 2.x (коды 01h–12h), а данная функция выдает уточненный (расширенный) код.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 59h;
- в BX — значение 0.

Функция возвращает в регистрах следующие значения:

- в AX — расширенный код ошибки (см. табл. 6.1.);
- в BH — класс ошибки (табл. 6.5.);

- в BL — код рекомендуемого действия (табл. 6.6.);
- в CH — местоположение ошибки (табл. 6.7.);

ВНИМАНИЕ

В результате выполнения функции будут разрушены регистры CL, DX, SI, DI, BP, DS и ES, поэтому их содержимое перед вызовом данной функции следует сохранить.

Таблица 6.5. Значения кодов класса ошибок

Код класса	Значение
01h	Нехватка ресурсов (недостаточно места на диске или нет свободных каналов ввода-вывода)
02h	Доступ к файлу временно запрещен (с ним работает другая программа)
03h	Доступ запрещен — у программы нет необходимых полномочий для работы
04h	Ошибка системной программы
05h	Отказ оборудования
06h	Отказ системы (разрушен файл конфигурации)
07h	Ошибка прикладной программы
08h	Не найдено
09h	Неверный формат
0Ah	Заблокировано
0Bh	Ошибка носителя информации
0Ch	Уже существует
0Dh	Неизвестно

Таблица 6.6. Значения кодов рекомендуемых действий

Код действия	Рекомендуемая реакция на ошибку
01h	Повторить несколько раз и в случае неудачи предложить пользователю либо прервать операцию, либо проигнорировать ошибку
02h	Повторить несколько раз с задержкой между повторами и в случае неудачи предложить пользователю либо прервать операцию, либо проигнорировать ошибку
03h	Запрос пользователю для повторного ввода информации (обычно необходимо в случае некорректно заданного имени файла или диска)

продолжение ⇒

Таблица 6.6 (*продолжение*)

Код действия	Рекомендуемая реакция на ошибку
04h	Прервать программу после освобождения используемых ресурсов
05h	Немедленно прервать программу
06h	Игнорировать ошибку
07h	Повтор после вмешательства пользователя, который должен устранить причину ошибки

Таблица 6.7. Значения кодов местоположения ошибок

Код	Местоположение ошибки
01h	Неизвестно
02h	Диск
03h	Сеть
04h	Устройство последовательного доступа
05h	Память

Прерывание Int 21h, функция 5Ah: открыть существующий файл

Открывает файл для чтения, записи или дозаписи информации. Указатель при этом устанавливается в начало файла.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 5Ah;
- в CX — атрибуты файла (см. табл. 6.2);
- в DS:DX — указатель на путь в виде строки ASCIIZ (в конце строки должно находиться не менее 13 байт, заполненных нулями для получения сгенерированного имени).

В случае успешного завершения операции функция возвращает в регистре AX дескриптор файла, открытого для чтения и записи в режиме совместимости; DS:DX при этом будет указывать на путь, дополненный сгенерированным именем файла.

Возможные коды ошибки: 03h, 04h, 05h.

Прерывание Int 21h, функция 5Bh: создать новый файл

Создает файл для записи. От функции 3Ch отличается тем, что завершается ошибкой, если файл с таким именем уже существует.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 5Bh;
- в CX — атрибуты создаваемого файла (см. табл. 6.2);
- в DS:DX — указатель на имя файла в виде строки ASCIIZ.

В случае успешного завершения операции функция возвращает в регистре AX дескриптор файла.

Возможные коды ошибки: 03h, 04h, 05h, 50h.

Низкоуровневые дисковые функции DOS

К этой группе относятся функции, обеспечивающие непосредственную работу с секторами данных на логических дисках. Применяются они либо для ускорения ввода-вывода данных в системах управления реального времени, либо в специальных программах, предназначенных для восстановления удаленных файлов или исправления повреждений в файловой структуре диска.

Каждая из функций данной группы имеет собственный номер прерывания. При выполнении этих прерываний разрушается содержимое всех регистров общего назначения, сохраняется только содержимое сегментных регистров. После выполнения функции необходимо также извлечь из стека содержимое регистра флагов, помещенное туда при вызове прерывания.

В случае ошибки все функции этой группы устанавливают флаг CF и возвращают в регистре AL код ошибки (см. табл. 6.1), а в регистре AH — код состояния устройства (табл. 6.8).

Таблица 6.8. Коды состояния устройства

Код	Состояние
01h	Неверная команда
02h	Дефектная адресная метка
03h	Диск защищен от записи

продолжение ⇨

Таблица 6.8 (*продолжение*)

Код	Состояние
04h	Сектор не найден
08h	Сбой DMA
10h	Ошибка данных — неправильная контрольная сумма (CRC)
20h	Сбой контроллера
40h	Сбой при выполнении поиска
80h	Устройство не отвечает

Прерывание Int 25h: абсолютное чтение секторов из разделов малого объема

Прерывание используется для чтения секторов с диска по заданным логическим номерам. Объем раздела не должен превышать 32 Мбайт.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AL — код логического диска (0 — A:, 1 — B: и т. д.);
- в CX — число считываемых секторов;
- в DX — номер начального сектора;
- в DS:BX — адрес буфера для размещения данных.

В случае успешного выполнения операции в буфер будут записаны данные, прочитанные с диска.

Прерывание Int 25h, функция FFFFh: абсолютное чтение секторов из разделов большого объема

Прерывание используется для чтения секторов с диска по заданным логическим номерам. Читает из разделов, имеющих объем более 32 Мбайт.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AL — код логического диска (0 — A:, 1 — B: и т. д.);
- в CX — значение 0FFFFh;
- в DS:BX — адрес блока параметров операции абсолютного чтения (табл. 6.9).

В случае успешного выполнения операции в буфер, адрес которого указан в блоке параметров, будут записаны данные, прочитанные с диска.

Таблица 6.9. Блок параметров операции абсолютного чтения

Смещение	Размер	Описание
00h	DWORD	Номер сектора
04h	WORD	Число считываемых секторов
06h	DWORD	Дальний (far) указатель на буфер данных

Прерывание Int 26h: абсолютная запись секторов в разделы малого объема

Используется для записи секторов на диск по заданным логическим номерам. Объем раздела не должен превышать 32 Мбайт.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AL — код логического диска (0 — A:, 1 — B: и т. д.);
- в CX — число записываемых секторов;
- в DX — номер начального сектора;
- в DS:BX — адрес буфера, содержащего записываемые данные.

Прерывание Int 26h, функция FFFFh: абсолютная запись секторов в разделы большого объема

Используется для записи секторов на диск по заданным логическим номерам. Читает из разделов, имеющих объем более 32 Мбайт.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AL — код логического диска (0 — A:, 1 — B: и т. д.);
- в CX — значение 0FFFFh;
- в DS:BX — адрес блока параметров операции абсолютной записи (формат блока приведен в табл. 6.10).

Таблица 6.10. Блок параметров операции абсолютной записи

Смещение	Размер	Описание
00h	DWORD	Номер сектора
04h	WORD	Число записываемых секторов
06h	DWORD	Дальний (far) указатель на буфер данных

Примеры использования функций DOS

В прикладных программах чаще всего применяются функции записи и считывания файлов. Программа SaveRusFont, приведенная на листинге 6.1, записывает в текущий каталог диска текущий шрифт MS-DOS, полученный из памяти видеоконтроллера, в двоичном представлении (файл font0.fnt) и в виде инверсного изображения в формате BMP (файл font0.bmp). Она является упрощенным вариантом программы формирования изображения стандартных американского и русского шрифтов MS-DOS, которую я использовал для создания рис. 1.1 и 1.2, приведенных в начале главы 1 «Работа с клавиатурой».

Кроме описанных в предшествующих главах процедур общего назначения, в программе SaveRusFont используются приведенные в том же листинге 6.1 вспомогательные подпрограммы, выполняющие следующие функции:

- процедура GrabRusFont выполняет операцию считывания шрифта из памяти видеоконтроллера (она уже была описана в главе 4 «Видеоконтроллеры»);
- процедура ShowRusFont отображает шрифт на экран в стандартном 256-цветном режиме VGA с разрешением 320×200;
- процедура CreateBMPFile создает на диске файл и записывает в него заголовок формата BMP;
- процедура WriteRasterString служит для записи в файл очередной строки изображения в формате BMP RGB;
- процедура CloseBMPFile завершает операцию записи файла изображения на диск.

Листинг 6.1. Запись русского шрифта на диск в двоичном формате и в виде инверсного изображения в формате BMP RGB

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

SEGMENT sseg para stack 'STACK'
```

```
        DB 400h DUP(?)
ENDS

DATASEG
; Текстовые сообщения
Txt1 DB LIGHTCYAN,0,23
     DB "ЗАПИСЬ НА ДИСК РУССКОГО ШРИФТА DOS",0
     DB LIGHTGREEN,12,20
     DB "Запись шрифта в двоичный файл FONT0.FNT",0
     DB YELLOW,24,35,"Ждите ...",0
Txt2 DB LIGHTGREEN,11,20
     DB "Запись шрифта в файл BMP-типа FONT0.BMP",0
     DB LIGHTGREEN,13,18
     DB "Будет произведен переход в графический режим",0
     DB YELLOW,24,24,"Нажмите любую клавишу и ждите",0
Txt3 DB LIGHTGREEN,12,28,"Запись шрифта завершена",0
     DB YELLOW,24,29,"Нажмите любую клавишу",0

; Буфер для сохранения шрифта (16x256 байт)
Font8x16 DB 4096 DUP(?)
; Позиция отображаемого символа
FontString DW ? ;номер строки шрифта
FontColumn DW ? ;номер колонки шрифта
; Имя для файла BMP
BMPFileName DB 'font0.bmp',0 ;имя для двоичного файла
; Кодовый номер файла
FileHandle DW 0
; Заголовок файла BMP
BMPStruct DB 'BM'    ;сигнатура
          DD 153654 ;размер файла в байтах
          DW 0      ;резерв
          DW 0      ;резерв
          DD 54     ;смещение области данных
          DD 40     ;размер описателя изображения
          DD 320    ;ширина изображения в пикселах
          DD 160    ;высота изображения в пикселах
          DW 1      ;число битовых плоскостей
          DW 24     ;число битов на пиксел
          DD 0      ;метод сжатия
          DD 153600 ;размер изображения в байтах
          DD 0      ;разрешение по горизонтали
          DD 0      ;разрешение по вертикали
          DD 0      ;число цветов в изображении
          DD 0      ;число важных цветов изображения
; Одна строка изображения в BMP-формате
BMPData    DB 960 DUP(?)
; Номер выводимой строки изображения
CurrentString DW ?
ENDS

CODESEG
```

продолжение ⇨

Листинг 6.1 (*продолжение*)

```
PROC SaveRusFont
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Считать шрифт из видеопамяти
        call    GrabRusFont
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition

; Записать шрифт на диск в двоичном формате
        ; Вывести текстовые сообщения на экран
        MShowColorText 3,Txt1
        ; Записать шрифт в файл font0.fnt
        mov     DX,offset Font8x16
        call    WriteFontFile

; Предупредить пользователя о начале продолжительной
; по времени операции записи файла BMP-типа на диск
        call    ClearScreen
        MShowColorString Txt1
        MShowColorText 3,Txt2
        call    GetChar
; Установить видеорежии VGA 320x200, 256 цветов
        mov     AX,13h
        int     10h
; Для наглядности отобразить шрифт на экране
; (8 строк по 32 симвoла)
        call    ShowRusFont

; Создать файл font0.bmp и записать заголовок файла
        call    CreateBMPFile
; Вывести изображение на диск построчно
        ; Вывод начинается с НИЖНЕЙ строки рисунка
        mov     [CurrentString],160
        mov     AX,0A000h
        mov     ES,AX
; Цикл вывода строк
@@P0:   ; Вычислить координаты начала экранной строки
        mov     AX,320
        mov     DX,[CurrentString]
        dec     DX
        mul     DX
        ; В SI записать указатель на строку экрана
        mov     SI,AX
```

```
        ; В DI записать указатель на строку BMP
        mov     DI,offset BMPData
; Сформировать строку BMP с инверсией цвета точек
        mov     CX,320      ;счетчик пикселов
@@P1:   mov     AL,[ES:SI]
        cmp     AL,0
        je      @@P2        ;на экране черная точка?
        ; Записать в строку BMP черную точку
        mov     [word ptr DI],0
        add     DI,2
        mov     [byte ptr DI],0
        inc     DI
        jmp short @@P3
        ; Записать в строку BMP белую точку
@@P2:   mov     [word ptr DI],0FFFFh
        add     DI,2
        mov     [byte ptr DI],0FFh
        inc     DI
@@P3:   inc     SI
        loop    @@P1
        ; Записать строку BMP-файла на диск
        call    WriteRasterString
        ; Перейти на строку вверх
        dec     [CurrentString]
        jnz     @@P0
; Завершить запись BMP-файла
        call    CloseBMPFile

; Переустановить текстовый режим
        mov     ax,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        call    SetCursorPosition
; Вывести сообщение о завершении записи шрифта
        MShowColorString Txt1
        MShowColorText 2,Txt3
        call    GetChar
@@End:  ; Переустановить текстовый режим
        mov     ax,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP SaveRusFont

;*************************************************
;* СЧИТЫВАНИЕ "РУССКОГО" ШРИФТА ИЗ ВИДЕОКОНТРОЛЛЕРА *
;*************************************************
PROC GrabRusFont near
        pushad
```

продолжение ⇨

Листинг 6.1 (*продолжение*)

```
; Перепрограммировать синхронизатор
        cli
        mov     DX,3C4h
        ; Установить последовательную адресацию
        ; ячеек видеопамяти
        mov     AX,0704h
        out     DX,AX
        sti
; Перепрограммировать графический контроллер
        mov     DX,3CEh
        ; Выбрать для считывания плоскость 2
        mov     AX,0204h
        out     DX,AX
        ; Запретить четную-нечетную адресацию
        mov     AX,0005h
        out     DX,AX
        ; Установить окно доступа по адресу A0000h
        mov     AX,0006h
        out     DX,AX

; Скопировать шрифт в буфер Font8x16
        mov     AX,0A000h
        mov     ES,AX
        mov     SI,0
        mov     BX,offset Font8x16
        mov     DX,256
@@M0:   mov     CX,16
@@M1:   mov     AL,[ES:SI]
        mov     [BX],AL
        inc     BX
        inc     SI
        loop    @@M1
        add     SI,16
        dec     DX
        jnz     @@M0
        popad
        ret
ENDP GrabRusFont

;************************************************
;* ОТОБРАЗИТЬ ШРИФТ НА ЭКРАНЕ В РЕЖИМЕ 320 X 200 *
;************************************************
PROC ShowRusFont near
        pusha
        push    ES
        mov     AX,0A000h
        mov     ES,AX
        mov     SI,offset Font8x16
        xor     DI,DI
```

```
        mov     [FontString],0
        mov     DL,0    ;цвет фона символа
        mov     DH,15   ;цвет символа
@@m0:   mov     [FontColumn],0
        push    DI
@@m1:   ; Отобразить очередной символ
        mov     AH,16   ;число строк в маске символа
@@m2:   ; Dтобразить строку изображения символа
        mov     AL,[SI] ;загрузить очередной байт маски
        mov     CX,8
@@m3:   ; Вывести на экран очередную точку изображения
        rol     AL,1
        jc      @@m4
        mov     [ES:DI],DL
        jmp short @@m5
@@m4:   mov     [ES:DI],DH
@@m5:   inc     DI
        loop    @@m3
        inc     SI
        add     DI,320-8
        dec     AH
        jnz     @@m2
        sub     DI,320*16-8-2
        inc     [FontColumn]
        cmp     [FontColumn],32
        jb      @@m1
        pop     DI
        add     DI,320*(16+4)
        inc     [FontString]
        cmp     [FontString],8
        jb      @@m0
        pop     ES
        popa
        ret
ENDP ShowRusFont

;********************************************
;* ОТКРЫТЬ BMP-ФАЙЛ И ЗАПИСАТЬ ЕГD ЗАГDЛDВDК *
;********************************************
PROC CreateBMPFile NEAR
        pusha
        ; Создать файл для записи
        mov     AH,3Ch
        mov     CX,0
        mov     DX,offset BMPFileName
        int     21h
        mov     [FileHandle],AX
        ; Записать заголовок BMP-файла
        mov     BX,[FileHandle]
        mov     CX,54
        mov     DX,offset BMPStruct
```

продолжение ⇗

Листинг 6.1 (*продолжение*)

```
        mov     AH,40h
        int     21h
        popa
        ret
ENDP CreateBMPFile

;****************************************************
;* ЗАПИСАТЬ СТРОКУ РАСТРОВОГО ИЗОБРАЖЕНИЯ В BMP-ФАЙЛ *
;****************************************************
PROC WriteRasterString NEAR
        pusha
        ; Записать блок данных
        mov     BX,[FileHandle]
        mov     CX,960
        mov     DX,offset BMPData
        mov     AH,40h
        int     21h
        popa
        ret
ENDP WriteRasterString

;********************
;* ЗАКРЫТЬ BMP-ФАЙЛ *
;********************
PROC CloseBMPFile NEAR
        pusha
        ; Закрыть файл
        mov     AH,3Eh
        mov     BX,[FileHandle]
        int     21H
        popa
        ret
ENDP CloseBMPFile
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуры чтения/записи шрифта на диск
include "list6_02.inc"

END
```

Стандартный графический режим VGA с номером 13h используется в программе для большей наглядности. На самом деле изображение может быть сформировано в любом свободном участке оперативной памяти, и отображать его на экран совершенно не обязательно, хотя и полезно при отладке программы. Формирование и вывод в файл строк изображения выполняется поочередно. Это простой,

экономичный (с точки зрения использования оперативной памяти), но довольно неэффективный способ работы: запись на диск осуществляется крайне медленно. При формировании строки изображения цвет инвертируется (черный цвет заменяется белым и наоборот), а затем вместо кода цвета подставляется реальное значение яркостей цветовых компонент режима RGB: 0, 0, 0 для черного и 0FFh, 0FFh, 0FFh для белого цвета.

Процедура записи шрифта в файл в простом двоичном коде ReadFontFile и процедура считывания шрифта из файла WriteFontFile будут применяться в последующих примерах, поэтому они выделены в отдельный модуль, приведенный в листинге 6.2. Программа SaveRusFont использует только процедуру записи шрифта (листинг 6.1).

Процедуры из листинга 6.2 записывают и считывают файл целиком, за один этап. Такой способ гораздо эффективнее, чем работа по кусочкам, однако он не всегда возможен: размер файла передается через регистр CX и потому ограничен предельным значением 16-разрядного числа (65 535). Если необходимо при помощи функций DOS обрабатывать файл большого объема, то делать это неизбежно приходится по частям. Поскольку при работе с дисками минимальной адресуемой единицей для операционной системы является кластер, то размер считываемого или передаваемого за одну операцию участка файла с целью ускорения работы системы выравнивается на максимально возможный размер кластера (32 768 байт).

Листинг 6.2. Запись шрифта в файл и считывание шрифта из файла в двоичном формате

```
DATASEG
; Текстовые сообщения
WrErr DB 12,23
      DB "Ошибка при записи файла font0.fnt",0
RdErr DB 12,20
      DB,"Не удается открыть файл шрифта font0.fnt",0
; Имя для двоичного файла, содержащего шрифт
BinFileName DB 'font0.fnt',0
ENDS

CODESEG
;****************************************
;*   ЗАПИСАТЬ ШРИФТ В ДВОИЧНЫЙ ФАЙЛ    *
;* Параметры:                           *
;* DS:DX - указатель на массив шрифта. *
;****************************************
PROC WriteFontFile NEAR
```

продолжение ⇗

Листинг 6.2 (*продолжение*)

```
        pusha
        ; Создать файл для записи
        push    DX
        mov     AH,3Ch
        mov     CX,0    ;доступ без ограничений
        mov     DX,offset BinFileName ;имя файла
        int     21h
        jc      @@Err
        mov     BX,AX   ;запомнить Handle в BX
        pop     DX
        ; Записать данные в файл
        mov     CX,4096 ;размер файла в байтах
        mov     AH,40h
        int     21h
        jc      @@Err
        ; Закрыть файл
        mov     AH,3Eh
        int     21H
        jc      @@Err
        popa
        ret
; Аварийный выход - ошибка при записи файла
@@Err:  MFatalError WrErr
ENDP WriteFontFile

;****************************************
;*      ПРОЧИТАТЬ ШРИФТ ИЗ ФАЙЛА        *
;* Параметры:                           *
;* DS:DX - указатель на массив шрифта.  *
;****************************************
PROC ReadFontFile NEAR
        pusha
        ; Открыть файл для чтения
        push    DX
        mov     AH,3Dh
        mov     AL,0    ;доступ "только для чтения"
        mov     DX,offset BinFileName ;имя файла
        int     21h
        jc      @@Err
        mov     BX,AX   ;запомнить Handle в BX
        pop     DX
        ; Прочитать данные из файла
        mov     CX,4096 ;размер файла в байтах
        mov     AH,3Fh
        int     21h
        jc      @@Err
        ; Закрыть файл
        mov     AH,3Eh
        int     21H
```

```
        jc      @@Err
        popa
        ret
; Аварийный выход - ошибка при чтении файла
@@Err:  MFatalError RdErr
ENDP ReadFontFile
ENDS
```

ПРИМЕЧАНИЕ

Запускать пример из листинга 6.1 можно на любом АТ-совместимом компьютере как с жесткого, так и с гибкого диска. Запись выполняется медленно — операция занимает несколько секунд даже при использовании жесткого диска (слишком медленно работают функции DOS). При запуске программы с гибкого диска снимите защиту записи.

Программа PCX256FontImage, приведенная в листинге 6.3, считывает из файла font0.fnt шрифт в двоичном формате, формирует в оперативной памяти (без отображения на экране монитора) 256-цветное изображение, преобразует его в формат PCX по описанному в главе 4 «Видеоконтроллеры» алгоритму, присоединяет таблицу палитры текстового режима VGA и сохраняет полученный результат в файле font0.pcx. Программа использует вспомогательную процедуру CreateFontImage, формирующую цветное изображение шрифта путем увеличения на единицу кода цвета для каждого очередного символа, и процедуру WritePCXFile, предназначенную для записи изображения в файл.

Листинг 6.3. Запись изображения русского шрифта в PCX-файл

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

SEGMENT sseg para stack 'STACK'
        DB 400h DUP(?)
ENDS

DATASEG
; Текстовые сообщения
Txt0 DB LIGHTCYAN,0,18
```

продолжение ⇨

Листинг 6.3 (*продолжение*)

```
     DB "СОЗДАНИЕ ИЗОБРАЖЕНИЯ ШРИФТА В ВИДЕ PCX-ФАЙЛА",0
     DB YELLOW,24,35,"Ждите ...",0
Txt1 DB LIGHTGREEN,12,21
     DB "Изображение записано в файл font0.pcx",0
     DB YELLOW,24,29,"Нажмите любую клавишу",0
; Буфер для сохранения шрифта (16x256 байт)
FontBx16 DB 4096 DUP(?)
; Позиция отображаемого символа
FontString DW ? ;номер строки шрифта
FontColumn DW ? ;номер колонки шрифта
; Имя для файла PCX
FileName DB 'font0.pcx',0
; Кодовый номер открытого файла
FileHandle    DW ?
; Размер файла
FileSize      DW ?
; Номер выводимой строки изображения
CurrentString DW ?
; Ширина изображения
ImageLength   DW ?
; Высота изображения
ImageHeigth   DW ?
ENDS

; Буфер для создания цветного изображения шрифта
SEGMENT IMAGEBUF para public 'DATA'
        DB 0FFFFh DUP(?)
ENDS
; Буфер для дисковых операций
SEGMENT FILEBUF para public 'DATA'
        DB 0FFFFh DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC PCX256FontImage
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Прочитать шрифт из файла
```

```
        mov     DX,offset Font8x16
        call    ReadFontFile

; Вывести текстовые сообщения на экран
        MShowColorText 2,Txt0
; Создать цветное изображение шрифта
        call    CreateFontImage

; ЗАПИСЬ СЖАТЫЕХ ДАННЫХ В БУФЕР
        ; Настроить DS на видеопамять
        push    DS
        mov     AX,IMAGEBUF
        mov     DS,AX
        mov     SI,0
        ; Настроить ES на буфер файла
        mov     AX,FILEBUF
        mov     ES,AX
        ; Очистить буфер
        mov     DI,0
        mov     CX,B000h
        cld
        mov     AX,0
        rep     stosw
        ; Пропустить заголовок
        mov     DI,128 ;счетчик байтов (длины файла)
        ; Использовать BX как счетчик строк изображения
        mov     BX,200
; Начало строки
@@PCXStringStart:
        ; Использовать DX как счетчик пикселов по строке
        mov     DX,320
; Начало очередной группы точек
@@NextGroup:
        lodsb
        dec     DX
        jz      @@OneByte ;последняя точка строки
        cmp     AL,[DS:SI]
        jne     @@OneByte ;у следующей точки другой цвет
; Отрезок из одноцветных точек
        mov     CL,1
@@NextByte:
        inc     SI
        inc     CL
        dec     DX
        jz      @@EndOfSegment ;последняя точка строки
        cmp     AL,[DS:SI]
        jne     @@EndOfSegment ;последняя точка отрезка
        cmp     CL,63
        jb      @@NextByte
@@EndOfSegment:
        mov     AH,AL
```

продолжение ⇨

Листинг 6.3 (*продолжение*)

```
        mov     AL,CL
        or      AL,0C0h
        stosw
        jmp short @@EndGroup
; Одна отдельная точка
@@OneByte:
        cmp     AL,0C0h
        jae     @@Corr
        stosb
        jmp short @@EndGroup
@@Corr: mov     AH,AL
        mov     AL,0C1h
        stosw
@@EndGroup:
        cmp     DX,0
        jnz     @@NextGroup
        dec     BX
        jnz     @@PCXStringStart
        pop     DS

; Записать признак формата регистров ЦАП
        mov     AX,FILEBUF
        mov     ES,AX
        mov     [byte ptr ES:DI],0Ch
        inc     DI
; Прочитать палитру в буфер файла
        mov     AX,1017h
        mov     BX,0
        mov     CX,256
        mov     DX,DI
        int     10h
; Откорректировать палитру
        mov     CX,768
@@ShPl: shl     [byte ptr ES:DI],2
        inc     DI
        loop    @@ShPl
; Запомнить размер файла
        mov     [FileSize],DI
; Заполнить заголовок файла
        mov     [byte ptr ES:00h],0Ah ;сигнатура
        mov     [byte ptr ES:01h],5   ;версия PCX
        mov     [byte ptr ES:02h],1   ;уплотнение по RLE
        mov     [byte ptr ES:03h],8   ;битов на пиксел
        mov     [word ptr ES:04h],0   ;XMIN
        mov     [word ptr ES:06h],0   ;YMIN
        mov     [word ptr ES:08h],319 ;XMAX
        mov     [word ptr ES:0Ah],199 ;YMAX
        mov     [byte ptr ES:41h],1   ;одна плоскость
        mov     [word ptr ES:42h],320 ;байтов в строке
```

```
        mov     [word ptr ES:44h],1   ;цветное изобр.

; Записать файл на диск
        call    WritePCXFile

; Вывести сообщение о завершении записи
        MShowColorText 2,Txt1
        call    GetChar
@@End:  ; Переустановить текстовый режим
        mov     ax,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP PCX256FontImage

;*********************************************
;* ОТОБРАЗИТЬ ШРИФТ НА ЭКРАНЕ В РЕЖИМЕ 320X200 *
;*********************************************
PROC CreateFontImage near
        pusha
        push    ES
; Настроить ES:DI на область, предназначенную для
; создания изображения
        mov     AX,IMAGEBUF
        mov     ES,AX
        mov     SI,offset Font8x16
        xor     DI,DI
; Очистить область изображения
        mov     AX,0
        mov     CX,8000h
        rep     stosw
; Вернуть указатель в начало области
        xor     DI,DI
; Создать цветное изображение шрифта
        mov     [FontString],0
        mov     DL,0    ;цвет фона символа
        mov     DH,0    ;цвет символа
@@m0:   mov     [FontColumn],0
        push    DI
@@m1:   ; Отобразить очередной символ
        mov     AH,16   ;число строк (байтов) в маске
@@m2:   ; Отобразить очередную строку символа
        mov     AL,[SI] ;загрузить очередной байт маски
        mov     CX,8
@@m3:   ; Вывести на экран очередную точку
        ; изображения символа
        rol     AL,1
        jc      @@m4
        mov     [ES:DI],DL
        jmp short @@m5
```

продолжение ⇒

Листинг 6.3 (*продолжение*)

```
@@m4:   mov     [ES:DI],DH
@@m5:   inc     DI
        loop    @@m3 ;конец цикла по строке маски
        inc     SI
        add     DI,320-8
        dec     AH
        jnz     @@m2 ;конец цикла рисования символа
        inc     DH   ;выбрать следующий оттенок палитры
        sub     DI,320*16-8-2
        inc     [FontColumn]
        cmp     [FontColumn],32
        jb      @@m1 ;конец цикла по строке таблицы
        pop     DI
        add     DI,320*(16+4)
        inc     [FontString]
        cmp     [FontString],8
        jb      @@m0 ;конец цикла рисования изображения
        pop     ES
        popa
        ret
ENDP CreateFontImage

;*****************************
;* ЗАПИСАТЬ ИЗОБРАЖЕНИЕ В ФАЙЛ *
;*****************************
PROC WritePCXFile NEAR
        pusha
        ; Создать файл для записи
        mov     AH,3Ch
        mov     CX,0
        mov     DX,offset FileName
        int     21h
        mov     [FileHandle],AX
        ; Записать данные в файл
        mov     BX,[FileHandle]
        mov     CX,[FileSize]
        mov     DX,0
        push    DS
        mov     AX,FILEBUF
        mov     DS,AX
        mov     AH,40h
        int     21h
        pop     DS
        ; Закрыть файл
        mov     AH,3Eh
        mov     BX,[FileHandle]
        int     21H
        popa
        ret
```

```
ENDP WritePCXFile
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуры чтения/записи шрифта на диск
include "list6_02.inc"

END
```

ПРИМЕЧАНИЕ

Для запуска примера пригоден любой AT-совместимый компьютер; запуск можно производить как с жесткого, так и с гибкого диска; перед запуском программы lst_6_03.exe необходимо создать в текущей папке файл шрифта font0.fnt (при помощи программы lst_6_01.exe).

Программа ShowPCXFile, приведенная в листинге 6.4, считывает с диска 256-цветное изображение в формате PCX (файл font0.pcx), преобразует его в формат TrueColor32 и выводит на экран. Считывание файла осуществляется при помощи процедуры ReadPCXFile, а преобразование в формат TrueColor — при помощи процедуры PCX256inTrueColor32Mode.

Листинг 6.4. Вывод на экран 256-цветного PCX-файла в режиме TrueColor32

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Номер видеорежима заранее не известен
GraphicsMode equ 0
; Логическая ширина строки в пикселах
LogicalStringLength equ 1024
; Ширина экрана в пикселах
ScreenLength equ 640
; Высота экрана, строк
ScreenHeigth equ 480

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Текстовые сообщения
```

продолжение ⇨

Листинг 6.4 *(продолжение)*

```
Txt1 DB 0,9,"ВЫВОД НА ЭКРАН 256-ЦВЕТНОГО PCX-ФАЙЛА "
     DB "В РЕЖИМЕ TRUECOLOR32",0
     DB 29,29,"Нажмите любую клавишу",0
Err1 DB 12,25,"Не удается открыть файл шрифта",0
; Имя файла PCX
FileName      DB 'font0.pcx',0
; Кодовый номер открытого файла
FileHandle    DW ?
; Размер файла
FileSize      DW ?
; Номер выводимой строки изображения
CurrentString DW ?
; Ширина изображения
ImageLength   DW ?
; Высота изображения
ImageHeigth   DW ?
; Смещение таблицы палитры от начала PCX-файла
PaletteOffset DW ?
ENDS

SEGMENT sseg para stack 'STACK'
        DB 400h DUP(?)
ENDS

; Буфер для PCX-файла
SEGMENT FILEBUF para public  'DATA'
        DB 0FFFFh DUP(?)
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC ShowPCXFile
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; "Захватить" текстовый шрифт
        call    GrabRusFont
; Установить видеорежим
        call    SetTrueColor32
; Установить режим прямой адресации памяти
        call    Initialization

; Установить серый цвет фона
        mov     [dword ptr DefaultBackground],404040h
```

```
; Установить белый цвет текста
        mov     [dword ptr DefaultColor],0FFFFFFh
; Закрасить экран в заданный цвет фона
        mov     ECX,ScreenHeigth
        shl     ECX,10
        mov     EDI,[LinearVideoBuffer]
        mov     EAX,[DefaultBackground]
@@NPix: mov     [GS:EDI],EAX
        add     EDI,4
        dec     ECX
        jnz     @@NPix
; Отобразить текстовые сообщения
        MGShowText 2,Txt1
; Прочитать с диска и отобразить PCX-файл
        call    PCX256inTrueColor32Mode

; Установить текстовый режим
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP ShowPCXFile

;************************************
;* ЧИТАТЬ PCX-ФАЙЛ В БУФЕР FILEBUF *
;************************************
PROC ReadPCXFile NEAR
        pusha
; Очистить переменные
        mov     [FileHandle],0
        mov     [FileSize],0
; Открыть файл для чтения
        mov     AH,3Dh
        mov     AL,0
        mov     DX,offset FileName
        int     21h
        jc      @@Err
        mov     [FileHandle],AX   ;запомнить Handle
; Получить длину файла, установив указатель на его конец
        mov     AH,42h
        mov     AL,2
        mov     BX,[FileHandle]
        xor     CX,CX
        xor     DX,DX
        int     21h
        jc      @@Err
        cmp     DX,0
        jne     @@Err  ;размер файла превышает 64 кб
        ; Запомнить размер файла
```

продолжение ⇨

Листинг 6.4 (*продолжение*)

```
        mov     [FileSize],AX
; Вернуть указатель в начало файла
        mov     AH,42h
        mov     AL,0
        mov     BX,[FileHandle]
        xor     CX,CX
        xor     DX,DX
        int     21h
        jc      @@Err
; Прочитать файл в буфер FILEBUF
        mov     BX,[FileHandle]
        pusha
        push    DS
        mov     AX,FILEBUF
        mov     DS,AX
        xor     DX,DX
        mov     CX,B000h
        mov     AH,3Fh
        int     21H
        pop     DS
        popa
        jc      @@Err
; Закрыть файл
        mov     AH,3Eh
        mov     BX,[FileHandle]
        int     21H
        jc      @@Err
        popa
        ret
; Аварийный выход - ошибка при чтении файла
@@Err:  MFatalError Err1
ENDP ReadPCXFile

;*********************************************
;* ПОКАЗАТЬ PCX-ИЗОБРАЖЕНИЕ В РЕЖИМЕ TRUECOLOR *
;*********************************************
PROC PCX256inTrueColor32Mode NEAR
        pushad
; Прочитать файл
        call    ReadPCXFile
; Настроить пару регистров ES:SI на область изображения
        mov     AX,FILEBUF
        mov     ES,AX
        mov     SI,128
; Вычислить ширину и высоту изображения
        mov     AX,[ES:42h]
        mov     [ImageLength],AX
        mov     AX,[ES:0Ah]
        sub     AX,[ES:6]
```

```
        inc     AX
        mov     [ImageHeigth],AX
; Вычислить смещение палитры от начала файла
        mov     AX,[FileSize]
        sub     AX,768
        mov     [PaletteOffset],AX
; Настроить EDI на видеопамять
        ; Установить в качестве начальной строки
        ; изображений 64-ю строку растра
        mov     EDI,64
        shl     EDI,12
        ; Прибавить адрес видеобуфера
        add     EDI,[LinearVideoBuffer]

; Цикл по строкам изображения
@@StringStart:
        mov     DX,[ImageLength]
        push    EDI
@@NextDataByte:
        ; Занести в CX единицу
        mov     CX,1
        mov     AL,[ES:SI]
        inc     SI
        ; Проверить признак того, что байт является
        ; счетчиком пикселов
        test    AL,10000000b
        jz      @@LoadRealColor
        test    AL,1000000b
        jz      @@LoadRealColor
        ; Занести счетчик в CX
        and     AL,111111b
        mov     CL,AL
        xor     CH,CH
        ; Занести в AL код цвета
        mov     AL,[ES:SI]
        inc     SI
; Взять из таблицы палитры реальные значения
; цветовых компонент
@@LoadRealColor:
        ; Умножить код цвета на 3 и занести в BX
        xor     AH,AH
        mov     BX,AX
        shl     AX,1
        add     BX,AX
        ; Прибавить к BX смещение таблицы
        add     BX,[PaletteOffset]
        ; Занести в EAX цвет, при загрузке изменяя
        ; порядок компонент на обратный
        xor     EAX,EAX
        mov     AX,[ES:BX]
```

продолжение ⇨

Листинг 6.4 (*продолжение*)

```
        xchg    AL,AH
        shl     EAX,8
        mov     AL,[ES:BX+2]
; Перейти к следующему пикселу
@@NextPixel:
        mov     [GS:EDI],EAX
        add     EDI,4
        dec     DX
        jz      @@NextString
        loop    @@NextPixel
        jmp short @@NextDataByte
; Перейти к следующей строке
@@NextString:
        pop     EDI
        add     EDI,4*LogicalStringLength
        dec     [ImageHeigth]
        jnz     @@StringStart ;конец внешнего цикла

; Ожидать нажатия любой клавиши
        call    GetChar
        popad
        ret
ENDP PCX256inTrueColor32Mode
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить набор процедур общего назначения,
; предназначенных для установки графических
; видеорежимов и работы в них
include "list4_02.inc"
; Подключить набор процедур вывода текста,
; предназначенных для режимов TrueColor32
include "list4_07.inc"

END
```

ПРИМЕЧАНИЕ

Данный пример предъявляет к оборудованию повышенные требования: для запуска примера в компьютере должен быть установлен видеоконтроллер, имеющий VESA BIOS версии не ниже 2.0, способный работать с видеорежимами TrueColor32 и имеющий не менее 4 Мбайт памяти. Запуск программы lst_6_04.exe можно производить как с жесткого диска, так и с гибкого диска; перед запуском необходимо создать в текущем каталоге файл font0.pcx при помощи программы lst_6_03.exe.

В листинге 6.5 приведен более сложный пример, объединяющий в себе разнообразные приемы работы, описанные в предыдущих разделах. Программа FontEditor представляет собой упрощенную версию редактора растрового шрифта VGA 8×16. Программа использует описанные выше процедуры общего назначения (в том числе — для работы в режиме линейной адресации памяти), а также ряд вспомогательных функций:

- процедура ShowFontTable предназначена для отображения редактируемого шрифта на экран в виде таблицы кнопок (16×16), которая служит для выбора редактируемого символа и располагается в правом верхнем углу экрана;
- процедура ShowEditedChar отображает текущий символ шрифта в увеличенном в 16 раз масштабе в левой верхней части экрана;
- процедура DrawMouseCursor выводит на экран указатель мыши, предварительно сохраняя расположенный под ним фон;
- процедура DeleteMouseCursor стирает курсор мыши с экрана путем восстановления фона;
- процедура ShowNewMouseCursorPosition обеспечивает перерисовку курсора в новой позиции экрана и предназначена для совместной работы с драйвером мыши MS Mouse из листинга 5.1 (см. главу 5 «Работа с мышью»);
- процедура TestRegion позволяет определить, в какой области экрана находился курсор в момент нажатия одной из кнопок мыши;
- процедура DrawButtons обеспечивает перерисовку управляющих кнопок с отображением их состояния (нажатая кнопка подсвечивается);
- процедура CopyCharMask копирует маску редактируемого символа из массива шрифта в маску текущего символа;
- процедура SaveCharMask копирует результат редактирования из маски текущего символа в массив шрифта;
- процедура Wait01Sec обеспечивает задержку гашения нажатой управляющей кнопки примерно на 0,1 с.

После запуска программа-редактор считывает из текущего каталога файл шрифта font0.fnt и переходит в графический 256-цветный режим с линейной адресацией памяти. Чтобы выбрать символ в качестве текущего, нужно установить курсор на его позицию в таблице шрифта и щелкнуть левой кнопкой мыши. Символы в таблице расположены по порядку ASCII-кодов, слева направо, сверху вниз.

Чтобы изменить маску текущего символа, нужно выбрать его в качестве текущего (изображение маски символа после этого появится в окне редактирования), установить курсор на надпись Редактировать и щелкнуть левой кнопкой мыши. После этого надпись зажигается (считается, что нажата кнопка Редактировать).

Собственно редактирование маски осуществляется установкой курсора мыши на квадратик увеличенного изображения символа, который соответствует некоторой точке маски, и щелчком левой или правой кнопки мыши. Щелчок левой кнопки приводит к установке соответствующего бита маски в состояние 1 (квадратик зажигается), а щелчок правой — к сбросу бита в 0 (квадратик гаснет).

После завершения редактирования нужно установить курсор на надпись Запомнить и щелкнуть левой кнопкой мыши. Если при выборе символа или в процессе редактирования была допущена ошибка, то можно вместо надписи Запомнить выбрать надпись Отменить.

Чтобы сохранить текущее состояние шрифта на диске, щелкните левой кнопкой мыши на надписи Сохранить шрифт (в левом нижнем углу экрана). Для завершения работы с программой щелкните левой кнопкой мыши на надписи Выйти из программы (в правом нижнем углу экрана). Если программа находится в режиме редактирования символа, то кнопки меню Сохранить шрифт и Выйти из программы заблокированы (программа не реагирует на их нажатие).

Программа записывает результат редактирования в тот же самый файл font0.fnt, из которого берет исходный вариант шрифта. Поэтому, если отредактированный шрифт необходим для работы, скопируйте его в файл с другим именем, чтобы случайно не стереть с него информацию при запуске демонстрационных примеров из данного раздела.

Листинг 6.5. Программа для редактирования шрифта 8×16

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Режим 640x480, 256 цветов
GraphicsMode equ 4101h
; Логическая ширина экрана в пикселах
LogicalStringLength equ 1024
; Физическая ширина экрана в пикселах
ScreenLength  equ  640
; Высота экрана, строк
```

```
ScreenHeigth equ 480

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

DATASEG
; Редактируемый шрифт
EditedFont DB 4096 DUP(?)
; Редактируемый символ
EditedChar DB 16 DUP(?)
; Смещение редактируемого символа в шрифте
EdChOffset DW ?
; Номер строки "активного" символа
ActiveCharString DW ?
; Номер колонки "активного" символа
ActiveCharColumn DW ?
; Текстовые сообщения
Text DB 17,14,"Окно",0
     DB 18,9,"редактирования",0
     DB 25,36,"Для выбора символа в таблице "
     DB "используйте",0
     DB 26,46,"левую клавишу мыши",0
Errl DB 12,26,"Мышь MS Mouse не обнаружена",0
CharMenu DB 20,9," Редактировать ",0
         DB 22,9,"   Отменить    ",0
         DB 24,9,"   Запомнить   ",0
MainMenu DB 28,0," Сохранить ",0
         DB 29,0,"   шрифт   ",0
         DB 28,69," Выйти из  ",0
         DB 29,69," программы ",0
; Маска курсора мыши
MCurs DB  8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
      DB  8,15, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
      DB  0, 8,15,15, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
      DB  0, 8,15,15,15,15, 8, 8, 0, 0, 0, 0, 0, 0, 0, 0
      DB  0, 0, 8,15,15,15,15,15, 8, 8, 0, 0, 0, 0, 0, 0
      DB  0, 0, 8,15,15,15,15,15,15,15, 8, 8, 0, 0, 0, 0
      DB  0, 0, 0, 8,15,15,15,15,15,15,15,15, 8, 8, 0, 0
      DB  0, 0, 0, 8,15,15,15,15,15,15,15,15,15,15, 8, 8
      DB  0, 0, 0, 0, 8,15,15,15,15,15,15,15,15,15, 8, 0
      DB  0, 0, 0, 0, 8,15,15,15,15,15,15,15,15, 8, 0, 0
      DB  0, 0, 0, 0, 0, 8,15,15,15,15,15,15, 8, 0, 0, 0
      DB  0, 0, 0, 0, 0, 8,15,15,15,15,15,15,15, 8, 0, 0
      DB  0, 0, 0, 0, 0, 0, 8,15,15,15, 8,15,15,15, 8, 0
      DB  0, 0, 0, 0, 0, 0, 8,15,15, 8, 0, 8,15,15,15, 8
      DB  0, 0, 0, 0, 0, 0, 0, 8, 8, 0, 0, 0, 8,15, 8, 0
      DB  0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 0, 8, 0, 0
```

продолжение

Листинг 6.5 (*продолжение*)

```
; Область сохранения фона под курсором
MBack DB 256 DUP(?)
; Абсолютный адрес курсора
MouseAddres DD ?
; Координаты областей (порядок: x0,y0,x1,y1)
Region1 DW  64,  0,191,255 ;окно редактирования символа
Region2 DW 256,  0,639,383 ;таблица для выбора символа
Region3 DW  72,320,175,335 ;кнопка "Редактировать"
Region4 DW  72,352,175,367 ;кнопка "Отменить"
Region5 DW  72,384,175,399 ;кнопка "Запомнить"
Region6 DW   0,448,104,479 ;кнопка "Сохранить"
Region7 DW 536,448,639,479 ;кнопка "Выход"
; Признаки активности кнопок
Button3Flag DB 0
Button4Flag DB 0
Button5Flag DB 0
Button6Flag DB 0
Button7Flag DB 0
; Признак попадания курсора в заданную область
RegionFlag DB ?
; Флаг состояния программы (0 - выбор символа,
; 1 - редактирование символа)
ProgramStatus DB ?
ENDS

SEGMENT sseg para stack 'STACK'
        DB 400h DUP(?)
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC FontEditor
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим
        mov     AX,3
        int     10h
; Прочитать редактируемый шрифт из файла
        mov     DX,offset EditedFont
        call    ReadFontFile
; Произвести поиск мыши по последовательным портам
        call    MSMouseSearch
        cmp     [COMPortNum],1  ;номер порта больше 1?
        ja      @@MouseNotFound ;мышь не найдена
; Установить режим прямой адресации памяти
        call    Initialization
```

```
; "Захватить" текстовый шрифт
        call    GrabRusFont
; Установить видеорежим
        call    SetVESAVideoMode
; Обнулить позицию курсора мыши
        mov     [XCoordinate],0
        mov     [YCoordinate],0
        mov     [OldXCoordinate],0
        mov     [OldYCoordinate],0
; Отобразить текстовые сообщения
        ; Установить черный цвет фона
        mov     [DefaultBackground],BLACK
        ; Установить зеленый цвет текста
        mov     [DefaultColor],LIGHTGREEN
        MGShowText 4,Text
; Инициализировать переменные
        mov     [ActiveCharString],0
        mov     [ActiveCharColumn],0
        mov     [ProgramStatus],0
; Отобразить шрифт
        call    ShowFontTable
; Отобразить увеличенный символ
        call    CopyCharMask
        call    ShowEditedChar
; Отобразить кнопки
        call    DrawButtons
; Отобразить курсор мыши первый раз
        mov     [MouseCursorStatus],1
        call    DrawMouseCursor
; Установить новый обработчик прерывания
        call    SetMSMouseInterrupt

; Цикл обработки сигналов мыши
@@MWait:
        cmp     [ProgramStatus],0
        jne     @@Status1
; УРОВЕНЬ 0
        ; Левая кнопка нажата?
        test    [ButtonsStatus],10b
        jz      @@MWait
; Область кнопки "Выход"?
        mov     SI,offset Region7
        call    TestRegion
        cmp     [RegionFlag],1
        je      @@Exit
; Область кнопки "Сохранить"?
        mov     SI,offset Region6
        call    TestRegion
        cmp     [RegionFlag],0
        je      @@EdStart
```

продолжение ⇨

Листинг 6.5 (*продолжение*)

```
        ; Записать шрифт на диск
        mov     [MouseCursorStatus],0
        call    DeleteMouseCursor ;убрать курсор
        mov     [Button6Flag],1   ;"зажечь" кнопку
        call    DrawButtons       ;перерисовать кнопки
        mov     DX,offset EditedFont
        call    WriteFontFile     ;сохранить шрифт
        mov     [Button6Flag],0   ;"погасить" кнопку
        call    DrawButtons       ;перерисовать кнопки
        call    DrawMouseCursor   ;перерисовать курсор
        mov     [MouseCursorStatus],1
        jmp     @@MWait
@@EdStart:
; Область кнопки "Редактировать"?
        mov     SI,offset Region3
        call    TestRegion
        cmp     [RegionFlag],0
        je      @@SelectChar
        ; Начать редактирование символа
        mov     [ProgramStatus],1
        mov     [Button3Flag],1
        mov     [MouseCursorStatus],0
        call    DeleteMouseCursor ;убрать курсор
        call    DrawButtons       ;перерисовать кнопки
        call    DrawMouseCursor   ;перерисовать курсор
        mov     [MouseCursorStatus],1
        jmp     @@MWait
@@SelectChar:
; Область таблицы символов?
        mov     SI,offset Region2
        call    TestRegion
        cmp     [RegionFlag],0
        je      @@MWait
        ; Вычислить положение символа
        mov     AX,[XCoordinate]
        sub     AX,[Region2]
        xor     DX,DX
        mov     BX,24
        div     BX
        mov     [ActiveCharColumn],AX
        mov     AX,[YCoordinate]
        sub     AX,[Region2+2]
        xor     DX,DX
        mov     BX,24
        div     BX
        mov     [ActiveCharString],AX
        ; Перерисовать таблицу и символ
        mov     [MouseCursorStatus],0
        call    DeleteMouseCursor ;убрать курсор
```

```
        call    ShowFontTable      ;отобразить шрифт
        call    CopyCharMask       ;скопировать символ
        call    ShowEditedChar     ;отобразить символ
        call    DrawMouseCursor    ;перерисовать курсор
        mov     [MouseCursorStatus],1
        jmp     @@MWait

; УРОВЕНЬ 1 (редактирование)
@@Status1:
        ; Левая кнопка нажата?
        test    [ButtonsStatus],10b
        jz      @@RightButtonTest
; Область кнопки "Отмена"?
        mov     SI,offset Region4
        call    TestRegion
        cmp     [RegionFlag],0
        je      @@SaveChar
        ; Перерисовать таблицу и символ
        mov     [MouseCursorStatus],0
        call    DeleteMouseCursor ;убрать курсор
        mov     [Button3Flag],0   ;"погасить" кнопку
        mov     [Button4Flag],1   ;"зажечь" кнопку
        call    DrawButtons       ;перерисовать кнопки
        call    ShowFontTable     ;отобразить шрифт
        call    CopyCharMask      ;скопировать символ
        call    ShowEditedChar    ;отобразить символ
        call    Wait01Sec         ;задержка
        mov     [Button4Flag],0   ;"погасить" кнопку
        call    DrawButtons       ;перерисовать кнопки
        call    DrawMouseCursor   ;перерисовать курсор
        mov     [MouseCursorStatus],1
        mov     [ProgramStatus],0
        jmp     @@MWait
@@SaveChar:
; Область кнопки "Запомнить"?
        mov     SI,offset Region5
        call    TestRegion
        cmp     [RegionFlag],0
        je      @@EditChar_MLeft
        ; Запомнить символ
        call    SaveCharMask
        ; Перерисовать таблицу и символ
        mov     [MouseCursorStatus],0
        call    DeleteMouseCursor ;убрать курсор
        mov     [Button3Flag],0   ;"погасить" кнопку
        mov     [Button5Flag],1   ;"зажечь" кнопку
        call    DrawButtons       ;перерисовать кнопки
        call    ShowFontTable     ;отобразить шрифт
        call    CopyCharMask      ;скопировать символ
        call    ShowEditedChar    ;отобразить символ
```

продолжение ⇨

Листинг 6.5 (*продолжение*)

```
        call    Wait01Sec           ;задержка
        mov     [Button5Flag],0     ;"погасить" кнопку
        call    DrawButtons         ;перерисовать кнопки
        call    DrawMouseCursor     ;перерисовать курсор
        mov     [MouseCursorStatus],1
        mov     [ProgramStatus],0
        jmp     @@MWait
@@EditChar_MLeft:
        mov     SI,offset Region1
        call    TestRegion
        cmp     [RegionFlag],0
        jz      @@MWait
        ; Вычислить положение пиксела в маске символа
        mov     CX,[XCoordinate]
        sub     CX,[Region1]
        shr     CX,4 ;разделить на ширину клетки (16)
        mov     DI,[YCoordinate]
        sub     DI,[Region1+1]
        shr     DI,4 ;разделить на высоту клетки (16)
        add     DI,offset EditedChar
        ; Установить пиксел
        mov     AL,80h
        shr     AL,CL
        or      [DI],AL
        ; Перерисовать символ
        mov     [MouseCursorStatus],0
        call    DeleteMouseCursor ;убрать курсор
        call    ShowEditedChar      ;отобразить символ
        call    DrawMouseCursor     ;перерисовать курсор
        mov     [MouseCursorStatus],1
        jmp     @@MWait
@@RightButtonTest:
        ; Правая кнопка нажата?
        test    [ButtonsStatus],1
        jz      @@MWait
        mov     SI,offset Region1
        call    TestRegion
        cmp     [RegionFlag],0
        jz      @@MWait
        ; Вычислить положение пиксела в маске символа
        mov     CX,[XCoordinate]
        sub     CX,[Region1]
        shr     CX,4 ;разделить на ширину клетки (16)
        mov     DI,[YCoordinate]
        sub     DI,[Region1+1]
        shr     DI,4 ;разделить на высоту клетки (16)
        add     DI,offset EditedChar
        ; Установить пиксел
        mov     AL,80h
```

```
        shr     AL,CL
        not     AL
        and     [DI],AL
        ; Перерисовать символ
        mov     [MouseCursorStatus],0
        call    DeleteMouseCursor ;убрать курсор
        call    ShowEditedChar    ;отобразить символ
        call    DrawMouseCursor   ;перерисовать курсор
        mov     [MouseCursorStatus],1
        jmp     @@MWait

@@Exit: ; "Мугнуть" кнопкой "Выход"
        mov     [MouseCursorStatus],0
        call    DeleteMouseCursor ;убрать курсор
        mov     [Button7Flag],1   ;"зажечь" кнопку
        call    DrawButtons       ;перерисовать кнопки
        call    Wait01Sec         ;задержка
        mov     [Button6Flag],0   ;"погасить" кнопку
        call    DrawButtons       ;перерисовать кнопки
        ; Восстановить прежний обработчик прерывания
        call    RestoreOldMSMouseInterrupt
; Установить текстовый режим
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h

; Выдать сообщение "мышь не обнаружена"
@@MouseNotFound:
        MFatalError Err1
ENDP FontEditor

;****************************************
;*    ОТОБРАЗИТЬ ШРИФТ В ВИДЕ ТАБЛИЦЫ    *
;* Параметры передаются через переменные *
;* ActiveCharString и ActiveCharColumn   *
;****************************************
PROC ShowFontTable near
        pushad
        mov     SI,offset EditedFont
        mov     EDI,[LinearVideoBuffer]
        add     EDI,256
        mov     [FontString],0
; Отобразить очередную строку символов
@@m0:   mov     [FontColumn],0
; Отобразить очередной символ
@@m1:   mov     DL,8    ;цвет фона символа
        mov     DH,15   ;цвет символа
        mov     CX,[ActiveCharString]
```

продолжение ⇨

Листинг 6.5 (*продолжение*)

```
        cmp     [FontString],CX
        jne     @@k0
        mov     CX,[ActiveCharColumn]
        cmp     [FontColumn],CX
        jne     @@k0
        mov     DL,7    ;цвет фона "активного" символа
        mov     DH,15   ;цвет "активного" символа
        ; Оставить разделительную черную строки
@@k0:   add     EDI,LogicalStringLength+1
        ; Закрасить верхние 3 строки знакоместа
        ; цветом фона символа
        mov     AL,3
@@k1:   mov     CX,22
@@k2:   mov     [GS:EDI],DL
        inc     EDI
        loop    @@k2
        add     EDI,LogicalStringLength-22
        dec     AL
        jnz     @@k1

; Отобразить строку изображения символа
        mov     AH,16   ;число строк в маске символа
@@m2:   ; Закрасить фоном 7 точек слева от символа
        mov     CX,7
@@k3:   mov     [GS:EDI],DL
        inc     EDI
        loop    @@k3
        ; Загрузить очередной байт маски символа
        mov     AL,[SI]
        mov     CX,8
@@m3:   ; Вывести на экран очередную точку изображения символа
        rol     AL,1
        jc      @@m4
        mov     [byte ptr GS:EDI],DL
        jmp short @@m5
@@m4:   mov     [byte ptr GS:EDI],DH
@@m5:   inc     EDI
        loop    @@m3
        ; Закрасить фоном 7 точек справа от символа
        mov     CX,7
@@k4:   mov     [GS:EDI],DL
        inc     EDI
        loop    @@k4
        inc     SI
        add     EDI,LogicalStringLength-22
        dec     AH
        jnz     @@m2

        ; Закрасить нижние 3 строки знакоместа
```

```
        ; цветом фона
        mov     AL,3
@@k5:   mov     CX,22
@@k6:   mov     [GS:EDI],DL
        inc     EDI
        loop    @@k6
        add     EDI,LogicalStringLength-22
        dec     AL
        jnz     @@k5
        ; Оставить разделительную черную строку
        add     EDI,LogicalStringLength-1
; Перейти к следующему символу
        sub     EDI,LogicalStringLength*24-24
        inc     [FontColumn]
        cmp     [FontColumn],16
        jb      @@m1
; Перейти к следующей строке
        add     EDI,LogicalStringLength*24-24*16
        inc     [FontString]
        cmp     [FontString],16
        jb      @@m0
        popad
        ret
ENDP ShowFontTable

;**************************************************
;* ОТОБРАЗИТЬ СИМВОЛ В УВЕЛИЧЕННОМ МАСШТАБЕ (16:1) *
;* Номер отображаемого символа определяется       *
;* переменными ActiveCharString и ActiveCharColumn  *
;**************************************************
PROC ShowEditedChar near
        pushad
; Отобразить символ слева сверху
        mov     EDI,[LinearVideoBuffer]
        add     EDI,64
; Загрузить указатель на маску символа
        mov     SI,offset EditedChar
; Отобразить символ в масштабе 16:1 (размер точки
; символа - 16x16 точек экрана)
        mov     DX,16   ;высота маски символа в пикселах
@@m0:   ; Нарисовать светло-серую разделительную полоску
        mov     CX,32
@@m1:   mov     [dword ptr GS:EDI],07070707h
        add     EDI,4
        loop    @@m1
        add     EDI,LogicalStringLength-128

        ; Строку символа повторить 14 раз
        mov     AH,14
@@m2:   ; Отображаем строку символа
```

продолжение ⇨

Листинг 6.5 (*продолжение*)

```
        mov     AL,[SI] ;прочитать байт маски
        mov     CX,8    ;ширина маски символа в точках
@@m3:   rol     AL,1
        jc      @@m4
        ; Отобразить 16 точек темно-серого цвета
        mov     [dword ptr GS:EDI],08080807h
        mov     [dword ptr GS:EDI+4],08080808h
        mov     [dword ptr GS:EDI+8],08080808h
        mov     [dword ptr GS:EDI+12],07080808h
        jmp short @@m5
@@m4:   ; Отобразить 16 точек белого цвета
        mov     [dword ptr GS:EDI],0F0F0F07h
        mov     [dword ptr GS:EDI+4],0F0F0F0Fh
        mov     [dword ptr GS:EDI+8],0F0F0F0Fh
        mov     [dword ptr GS:EDI+12],070F0F0Fh
@@m5:   add     EDI,16
        loop    @@m3
        ; Перейти на следующую строку изображения символа
        add     EDI,LogicalStringLength-128
        dec     AH
        jnz     @@m2

        ; Нарисовать черную разделительную полоску
        mov     CX,32
@@m6:   mov     [dword ptr GS:EDI],07070707h
        add     EDI,4
        loop    @@m6
        add     EDI,LogicalStringLength-128
        ; Перейти на следующую строку маски символа
        inc     SI
        dec     DX
        jnz     @@m0
        popad
        ret
ENDP ShowEditedChar

;************************************
;* ВЫВЕСТИ ИЗОБРАЖЕНИЕ КУРСОРА МЫШИ *
;************************************
PROC DrawMouseCursor near
        pushad
; Вычислить адрес начальной точки для вывода маски
        ; Умножить длину строки на номер строки (Y)
        movzx   EDI,[YCoordinate]
        shl     EDI,10
        ; Прибавить номер колонки (X)
        movzx   EAX,[XCoordinate]
        add     EDI,EAX
        ; Прибавить адрес видеобуфера
```

```
        add     EDI,[LinearVideoBuffer]
        ; Запомнить экранный адрес курсора
        mov     [MouseAddres],EDI
        ; Записать адрес маски в индексный регистр
        mov     SI,offset MCurs
        ; Записать адрес области сохранения фона
        mov     BX,offset MBack
; Вывести изображение
        mov     DX,16           ;высота маски
@@M0:   ; Вывести очередную строку маски
        mov     CX,16           ;ширина маски
@@M1:   ; Сохранить точку фона
        mov     AL,[GS:EDI]
        mov     [BX],AL
        ; Проверить точку маски
        lodsb
        and     AL,AL           ;код цвета равен нулю?
        jz      @@M2            ;пропустить точку
        mov     [GS:EDI],AL     ;вывести точку
@@M2:   ; Перейти к следующей точке
        inc     EDI
        inc     BX
        loop    @@M1
        ; Перейти на следующую строку
        add     EDI,LogicalStringLength-16
        dec     DX
        jnz     @@M0
        popad
        ret
ENDP DrawMouseCursor

;************************************
;* СТЕРЕТЬ ИЗОБРАЖЕНИЕ КУРСОРА МЫШИ *
;************************************
PROC DeleteMouseCursor near
        pushad
; Записать в индексный регистр экранный адрес изображения
        mov     EDI,[MouseAddres]
        ; Записать адрес области сохранения
        ; фона в индексный регистр
        mov     SI,offset MBack
; Вывести исходное изображение
        mov     DX,16           ;высота маски
@@M0:   mov     CX,16           ;ширина маски
@@M1:   lodsb
        mov     [GS:EDI],AL     ;вывести точку
        inc     EDI
        loop    @@M1
        add     EDI,LogicalStringLength-16
        dec     DX
```

продолжение ⇨

Листинг 6.5 (*продолжение*)

```
        jnz     @@M0
        popad
        ret
ENDP DeleteMouseCursor

;************************************
;* ОТОБРАЗИТЬ НОВОЕ ПОЛОЖЕНИЕ КУРСОРА *
;************************************
PROC ShowNewMouseCursorPosition near
        call    DeleteMouseCursor
        call    DrawMouseCursor
        ret
ENDP ShowNewMouseCursorPosition

;**************************************************
;* ПРОВЕРИТЬ ПОПАДАНИЕ КУРСОРА В УКАЗАННУЮ ОБЛАСТЬ *
;* Параметры:                                     *
;* DS:SI - указатель на область.                  *
;**************************************************
PROC TestRegion near
        pusha
; Обнулить флаг попадания курсора в область
        mov     [RegionFlag],0
; Сравнить координаты курсора и области
        mov     AX,[XCoordinate]
        cmp     AX,[SI]
        jb      @@End
        cmp     AX,[SI+4]
        ja      @@End
        mov     AX,[YCoordinate]
        cmp     AX,[SI+2]
        jb      @@End
        cmp     AX,[SI+6]
        ja      @@End
        mov     [RegionFlag],1
@@End:  popa
        ret
ENDP TestRegion

;********************************
;* ОТОБРАЗИТЬ УПРАВЛЯЮЩИЕ КНОПКИ *
;********************************
PROC DrawButtons near
        pusha
; Вывести кнопки редактирования символа
        mov     SI,offset CharMenu
        mov     [DefaultBackground],BLUE
        mov     [DefaultColor],LIGHTGREY
        cmp     [Button3Flag],0 ;кнопка активна?
```

```
        je      @@b0
        mov     [DefaultBackground],LIGHTBLUE
        mov     [DefaultColor],YELLOW
@@b0:   call    GShowString
        mov     [DefaultBackground],BLUE
        mov     [DefaultColor],LIGHTGREY
        cmp     [Button4Flag],0 ;кнопка активна?
        je      @@b1
        mov     [DefaultBackground],LIGHTBLUE
        mov     [DefaultColor],YELLOW
@@b1:   call    GShowString
        mov     [DefaultBackground],BLUE
        mov     [DefaultColor],LIGHTGREY
        cmp     [Button5Flag],0 ;кнопка активна?
        je      @@b2
        mov     [DefaultBackground],LIGHTBLUE
        mov     [DefaultColor],YELLOW
@@b2:   call    GShowString
; Вывести кнопки сохранения шрифта и выхода
        mov     SI,offset MainMenu
        mov     [DefaultBackground],BLUE
        mov     [DefaultColor],LIGHTGREY
        cmp     [Button6Flag],0 ;кнопка активна?
        je      @@b3
        mov     [DefaultBackground],LIGHTBLUE
        mov     [DefaultColor],YELLOW
@@b3:   call    GShowString
        call    GShowString
        mov     [DefaultBackground],BLUE
        mov     [DefaultColor],LIGHTGREY
        cmp     [Button7Flag],0 ;кнопка активна?
        je      @@b4
        mov     [DefaultBackground],LIGHTBLUE
        mov     [DefaultColor],YELLOW
@@b4:   call    GShowString
        call    GShowString
        popa
        ret
ENDP DrawButtons

;****************************************************
;* СКОПИРОВАТЬ МАСКУ СИМВОЛА В ОБЛАСТЬ РЕДАКТИРОВАНИЯ *
;****************************************************
PROC CopyCharMask near
        pushad
; Вычислить положение маски символа в массиве шрифта
        ; Загрузить указатель на шрифт
        mov     SI,offset EditedFont
        ; Умножить номер строки на 16
        mov     AX,[ActiveCharString]
```

продолжение ⇨

Листинг 6.5 *(продолжение)*

```
        shl     AX,4
        ; Прибавить номер колонки
        add     AX,[ActiveCharColumn]
        ; Умножить на размер символа в байтах (на 16)
        shl     AX,4
        add     SI,AX
        ; Запомнить смещение символа в шрифте
        mov     [EdChOffset],SI
; Скопировать маску редактируемого символа
        mov     DI,offset EditedChar
        mov     EAX,[SI]
        mov     [DI],EAX
        mov     EAX,[SI+4]
        mov     [DI+4],EAX
        mov     EAX,[SI+8]
        mov     [DI+8],EAX
        mov     EAX,[SI+12]
        mov     [DI+12],EAX
        popad
        ret
ENDP CopyCharMask


;*********************************
;* ЗАПОМНИТЬ НОВУЮ МАСКУ СИМВОЛА *
;*********************************
PROC SaveCharMask near
        pushad
        mov     SI,[EdChOffset]
; Скопировать маску редактируемого символа
        mov     DI,offset EditedChar
        mov     EAX,[DI]
        mov     [SI],EAX
        mov     EAX,[DI+4]
        mov     [SI+4],EAX
        mov     EAX,[DI+8]
        mov     [SI+8],EAX
        mov     EAX,[DI+12]
        mov     [SI+12],EAX
        popad
        ret
ENDP SaveCharMask

;*******************
;* ЗАДЕРЖКА НА 0,1 С *
;*******************
PROC Wait01Sec near
        pushad
        push    ES
```

```
        ; Настроить ES на сегмент данных BIOS
        mov     AX,0
        mov     ES,AX
        ; Запомнить текущее время
        mov     EAX,[ES:046Ch]
        ; Ожидать 3 "тика"
        add     EAX,3
@@dT:   cmp     EAX,[ES:046Ch]
        ja      @@dT
        pop     ES
        popad
        ret
ENDP Wait01Sec
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить набор процедур общего назначения,
; предназначенных для установки графических
; видеорежимов и работы в них
include "list4_02.inc"
; Подключить набор процедур вывода текста,
; предназначенных для 256-цветных режимов
include "list4_03.inc"
; Подключить набор процедур для непосредственной
; работы с мышью типа MS Mouse
include "list5_01.inc"
; Подключить процедуры чтения и записи шрифта
; в файл в двоичном формате
include "list6_02.inc"

END
```

ПРИМЕЧАНИЕ

Программа редактирования шрифта предъявляет к аппаратуре следующие требования: процессор не ниже i486, видеоконтроллер должен иметь поддержку VESA BIOS 2.0.

Прерывания BIOS для работы с дисками на низком уровне

Функции BIOS были разработаны для дисков старого типа, использовавших режим адресации цилиндр-головка-сектор (CHS),

а современные жесткие диски работают в режиме линейной адресации (LBA). Хотя у современных дисков есть механизм эмуляции режима CHS, использование этого механизма нежелательно — большая часть дискового пространства может оказаться недоступной для работы из-за ограничений, накладываемых на число цилиндров и секторов диска форматом параметров функций BIOS.

Прерывания BIOS нужны программисту в тех случаях, когда невозможно использовать функции DOS: для работы с гибкими дисками нестандартного формата и для восстановления поврежденной файловой структуры или стертых файлов. В отличие от DOS, функции BIOS предназначены для работы с физическими дисками, а не с их разделами (логическими дисками) — не забудьте об этом, задавая номер диска.

Для вызова дисковых функций BIOS используется прерывание `Int 13h`. После выполнения операции:

- в случае успешного завершения флаг переноса `CF` сбрасывается, в регистр `AH` заносится значение 0;
- в случае неудачи флаг `CF` устанавливается, а в регистр `AH` заносится код состояния дисковода, возможные значения которого приведены в табл. 6.11.

ПРИМЕЧАНИЕ

При выполнении операций с гибкими дисками ошибки могут возникать из-за того, что в момент обращения к диску мотор дисковода выключен либо не успел разогнаться до номинальной скорости (функции BIOS не имеют встроенного контроля скорости вращения диска). Операции чтения, записи, поиска и форматирования для гибкого диска в случае ошибки следует повторить трижды, каждый раз выполняя сброс. Если ошибка не устранена, то нужно выдать пользователю предупреждение и прервать выполнение операции.

Таблица 6.11. Значения кодов состояния дисковода

Код	Состояние дисковода
00h	Успешное завершение операции, ошибок нет
01h	Недопустимый номер функции или параметр
02h	Не найден адресный маркер
03h	Диск защищен от записи[1]
04h	Сектор не найден
05h	Сброс в исходное состояние не выполнен[2]
06h	Произошла смена диска[1]

Код	Состояние дисковода
07h	Повреждена таблица параметров дисковода[2]
08h	Выход за границу ПДП[1]
09h	Попытка выполнить ПДП через границу 64 Кбайт
0Ah	Обнаружен дефектный сектор[2]
0Bh	Обнаружена дефектная дорожка[2]
0Ch	Нестандартный формат носителя или дорожки
0Dh	В команде форматирования задано недопустимое число секторов
0Eh	Обнаружена адресная метка контрольных данных[2]
0Fh	Уровень арбитража ПДП вышел из диапазона[2]
10h	Неисправимая ошибка при чтении (по контрольному коду ECC или CRC)
11h	Ошибка данных, исправленная по контрольному коду[2]
20h	Отказ контроллера
40h	Сбой при выполнении поиска
80h	Диск не отвечает (тайм-аут)
Aah	Дисковод не готов[2]
BBh	Неизвестная ошибка[2]
CCh	Ошибка при записи[2]
E0h	Ошибка регистра состояния[2]
FFh	Ошибка при выполнении операции опознавания[2]

Индекс 1 в таблице означает, что код относится только к гибким дискам; индекс 2 — только к жестким дискам.

Прерывание Int 13h, функция 00h: сброс дисковой системы

Функция предназначена для выполнения сброса (перевода в исходное состояние) контроллера диска и подключенных к нему дисководов. Ее необходимо вызывать после сбоев при выполнении операций чтения, записи, верификации или форматирования на гибком диске — перед повторением операции.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 00h;
- DL — номер дисковода (00h–7Fh — гибкий диск, 80h–FFh — жесткий диск).

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

ПРИМЕЧАНИЕ

При установленном бите 7 регистра DL выполняется общий сброс всех гибких и жестких дисков.

Прерывание Int 13h, функция 01h: определить текущее состояние дисковой системы

Функция возвращает код завершения последней операции, выполненной на указанном дисководе.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 01h;
- в DL — номер дисковода (00h–7Fh — гибкий диск, 80h–FFh — жесткий диск).

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

Прерывание Int 13h, функция 02h: читать сектор

Функция выполняет операцию считывания в заданную область оперативной памяти информации из одного или нескольких секторов диска.

ПРИМЕЧАНИЕ

При использовании жестких дисков старшие два бита 10-разрядного номера цилиндра помещаются в старшие два бита регистра CL.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 02h;
- в AL — число считываемых секторов (должно быть больше 0);
- в CH — номер цилиндра;

- в CL — номер начального сектора;
- в DH — номер головки;
- в DL — номер дисковода (00h–7Fh — гибкий диск, 80h–FFh — жесткий диск);
- в ES BX — указатель на адрес буфера, в который производится считывание информации.

После завершения операции функция возвращает в регистре AH состояние дисковой системы. В случае успешного завершения операции будет возвращена следующая информация:

- в AL — число прочитанных секторов;
- в буфере — прочитанная с диска информация.

Прерывание Int 13h, функция 03h: записать сектор

Функция переписывает данные из заданной области оперативной памяти в один или несколько указанных секторов диска.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 03h;
- в AL — число записываемых секторов (должно быть больше 0);
- в CH — номер цилиндра;
- в CL — номер начального сектора;
- в DH — номер головки;
- в DL — номер дисковода (00h–7Fh — гибкий диск, B0h–FFh — жесткий диск);
- в ES·BX — указатель на адрес буфера, из которого производится считывание информации.

ПРИМЕЧАНИЕ

При использовании жестких дисков старшие два бита 10-разрядного номера цилиндра помещаются в старшие два бита регистра CL.

После завершения операции функция возвращает в регистре AH состояние дисковой системы. В случае успешного завершения операции в AL будет находиться число записанных секторов.

Прерывание Int 13h, функция 04h: проверить правильность записи

Функция предназначена для контроля правильности выполнения записи данных на диск путем считывания данных и проверки контрольного кода CRC.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 04h;
- в AL — число проверяемых секторов (должно быть больше 0);
- в CH — номер цилиндра;
- в CL — номер начального сектора;
- в DH — номер головки;
- в DL — номер дисковода (00h–7Fh — гибкий диск, 80h–FFh — жесткий диск);
- в ES:BX — указатель на адрес буфера, в который производится считывание информации.

ПРИМЕЧАНИЕ

При использовании жестких дисков старшие два бита 10-разрядного номера цилиндра помещаются в старшие два бита регистра CL.

После завершения операции функция возвращает в регистре AH состояние дисковой системы, а в регистре AL — число проверенных секторов.

Прерывание Int 13h, функция 05h: форматировать дорожку гибкого диска

Функция предназначена для форматирования дорожек, то есть подготовки поверхности диска к выполнению операций чтения/записи данных.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 05h;
- в CH — номер цилиндра (дорожки);
- в DH — номер головки;
- в DL — номер дисковода (00h–7Fh);

- в ES:BX — указатель на адрес буфера, в котором содержится список адресных полей. Список состоит из 4-байтных элементов (формат которых описан в табл. 6.12) — по одному элементу на каждый создаваемый сектор.

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

ВНИМАНИЕ

Нельзя применять функцию форматирования при работе с жесткими дисками: она устарела, и ее использование может либо привести к повреждению поверхности диска (часть дорожек станет нечитаемой), либо вообще полностью вывести диск из строя.

Таблица 6.12. Формат элемента списка адресных полей

Смещение	Размер	Описание
00h	BYTE	Номер дорожки
01h	BYTE	Номер головки
02h	BYTE	Номер сектора
03h	BYTE	Размер сектора (байт): 0 — 128; 1 — 256; 2 — 512; 3 — 1024

Прерывание Int 13h, функция 08h: получить параметры дисковода

Функция предназначена для определения параметров дисководов.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 08h;
- в DL — номер дисковода (00h–7Fh — гибкий диск, 80h–FFh — жесткий диск).

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

В случае успешного завершения операции функция возвращает в регистрах следующую информацию:

- в BL — код типа дисковода (код выдается только для гибких дисков, возможные значения кода описаны в табл. 6.13);

- в CH — младшие 8 разрядов максимального номера цилиндра;
- в CL — максимальный номер сектора (разряды 0–5) и два старших бита максимального номера цилиндра (разряды 6–7);
- в DH — максимальный номер головки;
- в DL — общее число дисководов в системе;
- в ES:DI — указатель на таблицу параметров гибкого диска (выдается только для гибких дисков).

Таблица 6.13. Значения кодов типа дисковода

Код	Тип дисковода
1	5,25", 360 Кбайт, 40 дорожек
2	5,25", 1,2 Мбайт, 80 дорожек
3	3,5", 720 Кбайт, 80 дорожек
4	3,5", 1,44 Мбайт, 80 дорожек

Прерывание Int 13h, функция 0Dh: сброс контроллера жесткого диска

Функция выполняет инициализацию контроллера жесткого диска, не затрагивая (в отличие от функции 00h) контроллер гибких дисков. Головки заданного дисковода перемещаются при этом на нулевую дорожку.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 0Dh;
- в DL — номер жесткого диска (80h–FFh).

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

Вызывать данную функцию следует при возникновении ошибок в работе контроллера (см. табл. 6.11 кодов состояния системы).

Прерывание Int 13h, функция 10h: проверить готовность жесткого диска к работе

Функция определяет готовность диска к выполнению операций ввода-вывода.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 10h;
- в DL — номер жесткого диска (80h–FFh).

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

Прерывание Int 13h, функция 11h: рекалибровка жесткого диска

Функция выполняет перемещение головок заданного дисковода на нулевую дорожку.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 11h;
- в DL — номер жесткого диска (80h–FFh).

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

Вызывать данную функцию следует в случае возникновении сбоев в работе диска (см. табл. 6.11).

Прерывание Int 13h, функция 16h: проконтролировать смену гибкого диска

Функция предназначена для обеспечения целостности данных на диске, то есть для защиты прикладной программы от внезапной смены пользователем гибкого диска. В дисковых дупликаторах (специальных программах, предназначенных для копирования гибких дисков) и инсталляторах (программах, обеспечивающих установку на компьютер прикладных пакетов) данная функция, наоборот, обеспечивает замену уже использованного гибкого диска на следующий.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 16h;
- в DL — номер дисковода гибких дисков (00h–7Fh).

После завершения операции функция возвращает в регистре AH результат проверки (значение 0 — диск не менялся, значение 06h — диск был заменен).

Прерывание Int 13h, функция 18h: установить тип носителя для форматирования

Функция позволяет установить основные параметры, используемые при форматировании диска. Эту функцию необходимо вызвать перед началом работы с функцией 05h, чтобы система BIOS могла установить корректное значение скорости передачи данных для используемого дисковода.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 18h;
- в CH — число цилиндров;
- в CL — число секторов на дорожке;
- в DL — номер дисковода (00h–7Fh).

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

В случае успешного завершения операции функция возвращает в ES:DI указатель на таблицу параметров гибкого диска.

Векторы параметров дисководов

Несколько ячеек в таблице векторов прерываний хранят указатели на таблицы параметров дисководов (вместо адресов процедур обработки прерываний). Вектор 1Eh (то есть 32-разрядное слово, расположенное в оперативной памяти по адресу 0000:0078h) хранит таблицу параметров гибких дисков для дисковода (табл. 6.14). Векторы 41h (адрес 0000:0104h) и 46h (адрес 0000:0124h) хранят таблицы параметров жестких дисков (для диска 0 и диска 1 соответственно), формат которых показан в табл. 6.15.

Таблица 6.14. Формат таблицы параметров гибких дисков

Смещение	Размер	Описание
00h	BYTE	Первый байт спецификации: биты 0–3 — время разгрузки головок; биты 4–7 — длительность шага головок
01h	BYTE	Второй байт спецификации: бит 0 — флаг режима ПДП; биты 1–7 — время загрузки головок

Смещение	Размер	Описание
02h	BYTE	Задержка перед выключением мотора (в «тиках» системных часов)
03h	BYTE	Размер сектора (байт): 0 — 128; 1 — 256; 2 — 512; 3 — 1024
04h	BYTE	Число секторов на дорожке
05h	BYTE	Длина межсекторного промежутка для операций чтения и записи
06h	BYTE	Длина области данных
07h	BYTE	Длина межсекторного промежутка для операции форматирования
08h	BYTE	Байт-заполнитель для форматирования
09h	BYTE	Время установки головок в миллисекундах
0Ah	BYTE	Время запуска мотора в 1/8 с

Таблица 6.15. Формат таблицы параметров жесткого диска

Смещение	Размер	Описание
00h	WORD	Число цилиндров
02h	BYTE	Число головок
03h	WORD	Не используется (всегда 0)
05h	WORD	Номер начального цилиндра предкомпенсации
07h	BYTE	Максимальная длина блока коррекции ошибок ECC
08h	BYTE	Байт контроля: биты 0–2 — не используются (всегда 0); бит 3 — установлен, если число головок больше 8; бит 4 — не используется (всегда 0); бит 5 — установлен, если изготовитель разместил карту дефектов на цилиндре с номером «максимальный рабочий цилиндр + 1»; бит 6 — запрет повторного контроля ECC; бит 7 — запрет контроля ECC
09h	BYTE	Не используется (всегда 0)
0Ah	BYTE	Не используется (всегда 0)
0Bh	BYTE	Не используется (всегда 0)
0Ch	WORD	Номер цилиндра зоны парковки
0Eh	BYTE	Число секторов на дорожке
0Fh	BYTE	Зарезервировано

ВНИМАНИЕ

Значения в этих таблицах устанавливаются функциями BIOS в процессе работы. Во избежание сбоев в работе компьютера не рекомендуется что-либо изменять в них непосредственно. Информацию из таблиц можно только читать, причем для современных жестких дисков эта информация вообще бесполезна (они работают в режиме линейной адресации секторов).

Улучшенный дисковый сервис BIOS

Дополнительный набор функций для работы с жесткими дисками, в документации [64] именуемый как Enhanced Disk Drive Service (сокращенно EDD) или Extended Fixed Disk Service, предложен фирмой Phoenix Technologies, которая является ведущим разработчиком BIOS для AT-совместимых компьютеров. Этот набор был согласован с изготовителями оборудования и в настоящее время стал международным стандартом — входит в набор стандартов ATA/ATAPI комитета T13 ANSI.

ПРИМЕЧАНИЕ

Черновики различных версий спецификаций EDD и ATA/ATAPI можно свободно скачивать с Интернет-страницы комитета T13 ANSI www.t13.org.

Дополнительные функции позволяют решить ряд проблем, связанных с появлением новых моделей дисков, и обеспечить новые возможности BIOS:

- работу с дисками объемом более 528 Мбайт;
- работу более чем с двумя устройствами;
- новый метод доступа к данным, независимый от физической геометрии диска (то есть от реального числа цилиндров, головок и секторов на дорожках);
- поддержку новых методов передачи данных (Fast PIO, DMA и режима мультисекторной передачи).

Преодоление барьера в 528 Мбайт

Первоначально в режиме адресации CHS информация о физическом адресе сектора данных передавалась от контроллера к исполнительным устройствам дисковода напрямую, без каких-либо модификаций. Однако в результате быстрого увеличения емкости дисководов изготовители персональных компьютеров неожиданно

столкнулись с рядом проблем. Например, барьер в 528 Мбайт возник из-за несоответствия диапазонов значений номеров цилиндра, головки и сектора, которые используются для адресации секторов в интерфейсе IDE (ATA) и в стандартных функциях BIOS (табл. 6.16).

Таблица 6.16. Ограничения на диапазоны значений номеров цилиндра, головки и сектора

Параметр	Ограничение		
	BIOS	IDE	Общее
Максимальное число секторов на дорожке	63	255	63
Максимальное число головок	256	16	16
Максимальное число цилиндров	1024	65536	1024
Емкость диска	8,4 GB	136,9 GB	528 MB

ПРИМЕЧАНИЕ

Рассматривая табл. 6.16, следует учитывать, что вместо общепринятого в вычислительной технике определения понятий «килобайт», «мегабайт» и «гигабайт» (1 Кбайт = 1024 байт, 1 Мбайт = 1024 Кбайт, 1 Гбайт = 1024 Мбайт) изготовители дисков в рекламных целях стали использовать так называемые «десятичные мегабайты» (1 MB = 1 млн байт, 1 GB = 1 млрд байт). В «десятичных мегабайтах» объем диска, естественно, получается больше.

Чтобы решить проблему адресации секторов и согласовать между собой диапазоны значений параметров BIOS и IDE, изготовители оборудования стали использовать метод «измененной геометрии диска». В этом случае, во-первых, контроллер диска выдает не реальные, а преобразованные координаты сектора на диске, а во-вторых, функции BIOS также могут выполнять преобразование координат. Такое решение, однако, породило новые проблемы: некоторые прикладные программы работают с дисками напрямую, и при использовании «измененной геометрии» может возникнуть путаница, приводящая к потере информации на диске.

Фирма Phoenix Technologies предложила использовать для создания измененной геометрии диска метод битового сдвига. Это простой метод трансляции адресов секторов, при котором BIOS манипулирует только двумя параметрами геометрии диска: числом цилиндров и числом головок. Режим трансляции, устанавливаемый для диска при использовании алгоритма Phoenix, определяется числом цилиндров в реальной (физической) геометрии (табл. 6.17). Преимущество метода Phoenix состоит в том, что он позволяет работать с любыми дисками, в том числе со старыми моделями, в которых

не реализован режим линейной адресации секторов. Недостатком метода является наличие определенных ограничений на физическую геометрию диска: как видно из таблицы, у дисков с большим числом цилиндров должно быть больше четырех головок.

Таблица 6.17. Трансляция адресов секторов методом битового сдвига фирмы Phoenix

Физические параметры диска		Преобразование координат		Максимальный объем диска
Количество цилиндров	Количество головок	Номер цилиндра	Номер головки	
1 < C ≤ 1024	H ≤ 16	C' = C	H' = H	528 MB
1024 < C ≤ 2048	H ≤ 16	C' = C/2	H' = 2H	1 GB
2048 < C ≤ 4096	H ≤ 16	C' = C/4	H' = 4H	2,1 GB
4096 < C ≤ 8192	H ≤ 16	C' = C/8	H' = 8H	4,2 GB
8192 < C ≤ 16384	H ≤ 16	C' = C/16	H' = 16H	8,4 GB
16384 < C ≤ 32768	H ≤ 8	C' = C/32	H' = 32H	8,4 GB
32768 < C J 65536	H ≤ 4	C' = C/64	H' = 64H	8,4 GB

ВНИМАНИЕ

В современных моделях дисков вместо режима CHS с измененной геометрией используется режим линейной адресации секторов LBA, полностью скрывающий от пользователя реальную (физическую) геометрию диска. Преобразование линейного адреса в мнимый CHS-адрес в этом случае полностью возлагается на BIOS.

Для дисков, поддерживающих LBA, разработчики BIOS часто используют алгоритм трансляции с фиксированным числом секторов в измененной конфигурации диска. При этом считается, что при любом объеме диска на дорожке 63 сектора, а число головок и цилиндров определяется по правилам, приведенным в табл. 6.18.

Таблица 6.18. Трансляция адресов при фиксированном числе сектороа в измененной геометрии диска

Объем диска	Параметры измененной геометрии		
	Число секторов на дорожке	Число головок	Число цилиндров
X ≤ 528 MB	63	16	X / (63 * 16 * 512)
528 MB< X ≤ 1 GB	63	32	X / (63 * 32 * 512)

Объем диска	Параметры измененной геометрии		
	Число секторов на дорожке	Число головок	Число цилиндров
1 GB < X ≤ 2,1 GB	63	64	X / (63 * 64 * 512)
2,1 GB < X ≤ 4,2 GB	63	128	X / (63 * 128 * 512)
4,2 GB < X ≤ 8,4 GB	63	256	X / (63 * 256 * 512)

В том случае, если при выполнении процедуры самотестирования системы (после сброса или включения питания) диск при опросе выдает для режима CHS значение 63 сектора на дорожке, результаты трансляции при использовании обоих описанных методов совпадают. Если число секторов, сообщаемое диском, меньше 63, возникают различия: при использовании метода Phoenix в процедурах BIOS используется значение, полученное от диска, а алгоритм, ориентированный на режим LBA, всегда устанавливает в измененной геометрии 63 сектора на дорожке.

ВНИМАНИЕ

Если перенос информации с одного компьютера на другой осуществляется при помощи жесткого диска, в BIOS обоих компьютеров следует устанавливать для этого диска режим LBA.

Таблицы параметров диска

В таблице векторов прерываний имеются указатели на таблицы параметров для дисков 0 (вектор 41h, адрес 0000:0104h) и 1 (вектор 46h, адрес 0000:0124h). Значения в этих таблицах устанавливаются функциями BIOS в процессе начального тестирования системы после включения питания или сброса.

Структура стандартной таблицы параметров жесткого диска (Fixed Disk Parameter Table, сокращенно FDPT) описана в табл. 6.19.

Таблица 6.19. Формат стандартной таблицы параметров жесткого диска FDPT

Смещение	Размер поля	Назначение поля
00h	WORD	Физическое количество цилиндров
02h	BYTE	Физическое количество головок
03h	BYTE	Признак нетранслированной таблицы (имеет значение 0)

продолжение ⇗

Таблица 6.19 (*продолжение*)

Смещение	Размер поля	Назначение поля
04h	BYTE	Зарезервировано (имеет значение 0)
05h	WORD	Номер начального цилиндра предкомпенсации (устаревший параметр; имеет значение 0)
07h	BYTE	Зарезервировано (имеет значение 0)
08h	BYTE	Байт управления: биты 0–2 — не используются (имеют значение 0); бит 3 — установлен, если число головок больше 8; бит 4 — не испольэуются (имеет значение 0), бит 5 — установлен, если изготовитель разместил карту дефектов на цилиндре с номером «максимальный рабочий цилиндр + 1»; бит 6 — запрет повторного контроля ECC; бит 7 — запрет контроля ECC
09h	WORD	Зарезервировано (имеет значение 0)
0Bh	BYTE	Зарезервировано (имеет эначение 0)
0Ch	WORD	Номер цилиндра зоны парковки головок (устаревший параметр: обычно не используется и имеет значение 0)
0Eh	BYTE	Число секторов на дорожке
0Fh	BYTE	Зарезервировано

ВНИМАНИЕ

Информацию из таблиц FDPT можно только читать — во избежание разрушения информации на дисках категорически не рекомендуется что-либо в них изменять.

Таблица параметров жесткого диска, полученная после преобразования геометрии диска, называется транслированной (Translated Fixed Disk Parameter Table, сокращенно **TFDPT**). Она имеет такой же размер и такое же количество полей, как и стандартная таблица **FDPT**, но назначение некоторых полей отличается (табл. 6.20).

Таблица 6.20. Формат транслированной таблицы параметров жесткого диска TFDPT

Смещение	Размер поля	Назначение поля
00h	WORD	Логическое количество цилиндров (не более 1024)
02h	BYTE	Логическое количество головок (не более 256)
03h	BYTE	Сигнатура транслированной таблицы (имеет значение Axh)
04h	BYTE	Физическое количество секторов на дорожке (не более 63)
05h	WORD	Номер начального цилиндра предкомпенсации (устаревший параметр, имеет значение 0)
07h	BYTE	Зарезервировано (имеет значение 0)
08h	BYTE	Байт управления:
		биты 0–2 — не используются (имеют значение 0);
		бит 3 — установлен, если число головок больше 8;
		бит 4 — не используются (имеет значение 0);
		бит 5 — установлен, если изготовитель разместил карту дефектов на цилиндре с номером «максимальный рабочий цилиндр + 1»;бит 6 — запрет повторного контроля ECC;
		бит 7 — запрет контроля ECC
09h	WORD	Физическое количество цилиндров (не более 65536)
0Bh	BYTE	Физическое количество головок (не более 16)
0Ch	WORD	Номер цилиндра зоны парковки (устаревший параметр: обычно не используется и имеет значение 0)
0Eh	BYTE	Логическое количество секторов на дорожке
0Fh	BYTE	Контрольная сумма

Тип таблицы, который используется для диска, определяется физическим количеством имеющихся на нем цилиндров: если число цилиндров меньше либо равно 1024, используется стандартная FDPT,

а если больше — TFDPT. Как уже было указано выше, прикладным программам доступны только таблицы дисков 0 и 1 (то есть устройств с номерами 80h и 81h).

Физические значения параметров в таблицах FDPT и TFDPT устанавливаются BIOS в соответствии со значениями, возвращаемыми дисками в ответ на команду идентификации ID DRIVE, которую мы будем подробно рассматривать в главе, посвященной интерфейсу ATA/ATAPI. Предельные значения физических параметров в стандартной FDPT: 1024 цилиндра, 16 головок, 63 сектора; В TFDPT предельные значения физических параметров несколько иные: 65536 цилиндров, 16 головок, 63 сектора. Логические значения параметров используются только в режиме трансляции адресов секторов и встречаются только в TFDPT; предельные значения логических параметров: 1024 цилиндра, 256 головок, 63 сектора.

Поля «Номер начального цилиндра предкомпенсации» и «Номер цилиндра зоны парковки» устарели и были внесены в стандарт Phoenix только с целью совместимости с самыми старыми моделями дисков, которые в настоящий момент уже вообще вышли из употребления. Эти поля *всегда* содержат значение 0.

ВНИМАНИЕ

Термин «физическое/логическое количество цилиндров/головок» используется в таблицах FDPT и TFDPT некорректно: на самом деле соответствующие поля таблицы содержат *максимальный номер* цилиндра или головки, причем счет номеров ведется с *нуля*.

Для обслуживания прикладных программ, работающих с жесткими дисками напрямую (в обход прерывания Int 13h), стандарт Phoenix предусматривает специальную функцию (функцию 48h прерывания Int 13h), которая выдает более детальную информацию о дисках 0–3 (то есть устройствах с номерами 80h-83h). Информация об устройстве выдается в виде таблицы, которая называется дополнением FDPT и обозначается в документации как Device Parameter Table Extension (**DPTE**). Формат DPTE показан в табл. 6.21.

Таблица 6.21. Формат таблицы DPTE

Смещение	Размер поля	Назначение поля
00h	WORD	Базовый адрес блока регистров команды
02h	WORD	Базовый адрес блока регистров управления

Смещение	Размер поля	Назначение поля
04h	BYTE	Старший полубайт регистра номера головки: биты 0–3 — не используются (имеют значение 0); бит 4 — бит номера устройства ATA (0 — ведущий диск, 1 — ведомый диск); бит 5 — зарезервирован (имеет значение 1); бит 6 — режим адресации (0 — CHS, 1 — LBA); бит 7 — зарезервирован (имеет значение 1)
05h	BYTE	Используется для нужд BIOS
06h	BYTE	Информация об IRQ: биты 0–3 — номер IRQ для данного диска; биты 4–7 — не используются (имеют значение 0)
07h	BYTE	Счетчик секторов для режима мультисекторной передачи
08h	BYTE	Информация о DMA: биты 0–3 — номер канала DMA для данного диска; биты 4–7 — номер используемого режима DMA
09h	BYTE	Информация об IRQ: биты 0–3 — номер используемого режима PIO; биты 4–7 — не используются (имеют значение 0)
0Ah	WORD	Флаги параметров аппаратного обеспечения (свойство присутствует, если соответствующий флаг имеет значение 1): бит 0 — система сконфигурирована для режима Fast PIO; бит 1 — система сконфигурирована для режима Fast DMA; бит 2 — разрешены мультисекторные операции; бит 3 — в режиме CHS используется трансляция адресов; бит 4 — система сконфигурирована для режима LBA;

продолжение ⇨

Таблица 6.21 (*продолжение*)

Смещение	Размер поля	Назначение поля
		бит 5 — устройство работает со сменными носителями;
		бит 6 — устройство использует интерфейс ATAPI;
		бит 7 — используется 32-разрядный режим передачи данных;
		бит 8 — устройство ATAPI использует DRQ по прерыванию;
		биты 9–10 — код метода трансляции адресов в режиме CHS;
		бит 11 — система сконфигурирована для режима Ultra DMA;
		биты 12–15 — зарезервированы (имеют значение 0)
0Ch	WORD	Зарезервировано (имеет значение 0)
0Eh	BYTE	Номер версии «расширения» FDPT (имеет значение 11h)
0Fh	BYTE	Контрольная сумма (двоичное дополнение суммы байтов 00h–0Eh)

ВНИМАНИЕ

Данные из DPTE можно *только считывать* (информация в таблице защищена контрольной суммой, попытка перезаписи значения в одном из полей приведет к перезагрузке компьютера). Последствия перезаписи всей таблицы (с перерасчетом контрольной суммы) могут быть более печальными: информация на диске окажется разрушенной вследствие неправильной адресации секторов.

Рассмотрим поля таблицы DPTE более подробно. Слово со смещением 00h содержит физический базовый адрес блока регистров команды интерфейса ATA (ATA Command Block Registers), который в документации может также именоваться базой портов ввода-вывода (I/O Port Base). Слово со смещением 02h содержит физический базовый адрес блока регистров управления интерфейса ATA (ATA Control Block Registers), который в документации может также именоваться базой управляющих портов (Control Port Base). Сведения о физических адресах регистров контроллера ATA необходимы прикладным программам, которые непосредственно (в обход прерываний операционной системы) работают с контроллером дисков.

Байт со смещением 04h содержит так называемый «Префикс головки» (Head Prefix) — старший полубайт для регистра номера головки контроллера ATA. Префикс головки определяет, к какому диску канала относится данная таблица (если бит 4 равен 0 — ведущий диск, если 1 — ведомый), а также какой способ адресации секторов используется при работе с этим диском (если бит 6 равен 0 — используется режим CHS, если 1 — LBA). Перед записью в регистр номера головки ATA прикладная программа должна выполнить операцию логического сложения (логического ИЛИ) содержимого данного байта со значением номера головки.

Байт со смещением 05h предназначен для «внутренних нужд» BIOS. Он хранит значение количества сдвигов, необходимых для преобразования физического числа цилиндров в логическое.

Байт со смещением 06h хранит номер IRQ канала ATA, к которому подключен диск, описываемый данной таблицей. Данная информация необходима программам, непосредственно работающим с жесткими дисками.

Байт со смещением 07h содержит значение размера блока данных в секторах, который используется BIOS в режиме мультисекторной передачи данных, если диск запрограммирован на подобный режим работы. Если режим мультисекторной передачи не используется для данного диска, то значение байта равно нулю.

Байт со смещением 08h содержит в младшем полубайте номер канала DMA, используемого каналом ATA, к которому подключен диск, а в старшем полубайте — номер заданного для диска режима DMA. Содержимое данного байта имеет значение только в том случае, если используется Fast или Ultra DMA, то есть если установлен бит 1 или бит 11 слова опций аппаратного обеспечения; в случае, если диск запрограммирован на работу в режиме PIO, значение данного байта следует игнорировать. Учтите, что при работе с диском используется *самый быстрый* из возможных для него режимов DMA: номер DMA не ограничен значением 2, возвращаемым диском по команде идентификации устройства.

Байт со смещением 09h содержит номер режима Fast PIO, на который запрограммирован данный диск, если подобный режим используется вместо обычного PIO; в противном случае значение байта равно нулю.

Слово со смещением 0Ah содержит флаги опций аппаратного обеспечения: если какой-либо флаг установлен в 1, то у диска имеется соответствующее данному флагу свойство. Если слово флагов имеет

значение 0 (все флаги в «сброшенном» состоянии), то диск относится к устаревшему типу, имеет число цилиндров менее 1024 и работает в стандартном (самом медленном) режиме PIO. Рассмотрим подробно отдельные разряды слова флагов.

- Бит 0 принимает значение 1, если система сконфигурирована для работы в режиме Fast PIO. Если флаг имеет значение 1, то для конфигурирования системы используется байт 9 DPTE; если флаг сброшен в 0, то байт 9 DPTE игнорируется.
- Бит 1 принимает значение 1, если система сконфигурирована для работы в режиме Fast DMA. Если флаг имеет значение 1, то для конфигурирования системы используется байт 8 DPTE.
- Бит 2 принимает значение 1, если система сконфигурирована для работы в режиме мультисекторной передачи (Block PIO). Если флаг имеет значение 1, то число секторов, передаваемых в мультисекторном режиме, задается счетчиком секторов — байтом 7 DPTE; если флаг сброшен в 0, то байт 7 DPTE игнорируется.
- Бит 3 принимает значение 1, если диск содержит более 1024 цилиндров, то есть если в режиме CHS используется трансляция адресов. Когда данный флаг установлен, BIOS использует значение байт 5 DPTE при трансляции адресов секторов.
- Бит 4 принимает значение 1, если диск сконфигурирован для работы в режиме LBA. Режим LBA может быть задан даже в том случае, если диск имеет менее 1024 цилиндров, то есть значение данного флага устанавливается независимо от значения бита 3. Следует учитывать, что режим LBA используется только «дополнительными» функциями прерывания Int 13h с номерами 41h-48h, а стандартные функции всегда работают в режиме CHS.
- Бит 5 принимает значение 1, если устройство работает со сменными носителями информации.
- Бит 6 принимает значение 1, если устройство использует пакетный интерфейс ATAPI.
- Бит 7 принимает значение 1, если система использует при работе с диском 32-разрядный режим передачи данных.
- Бит 8 принимает значение 1, если устройство ATAPI вырабатывает сигнал прерывания, когда готово к передаче пакета; если флаг сброшен в ноль, то готовность устройства к передаче пакета определяется по состоянию сигнала DRQ. Состояние данного флага имеет значение только в том случае, если установлен бит 6

(устройство использует интерфейс ATAPI); если бит 6 сброшен, значение бита 8 игнорируется.

- Биты 9–10 содержат код метода трансляции адресов, используемого в режиме CHS: 00b — метод битового сдвига фирмы Phoenix, 01b — LBA-совместимый режим, 10b — зарезервированное значение, 11b — собственный алгоритм трансляции разработчика BIOS. Если бит 3 сброшен в ноль, значение данного поля игнорируется.
- Бит 11 принимает значение 1, если система сконфигурирована для работы в режиме Ultra DMA. Если флаг имеет значение 1, то для конфигурирования системы используется байт 8 DPTE.
- Биты 12–15 — зарезервированы и должны иметь значение 0.

Слово со смещением 0Ch зарезервировано. Оно должно содержать значение 0.

Байт со смещением 0Eh содержит номер версии стандарта Phoenix, которому соответствует формат таблицы дополнения FDPT, записанный в двоично-десятичном коде BCD. Значение данного байта должно быть равно 11h.

Байт со смещением 0Fh содержит контрольную сумму — двоичное дополнение суммы байтов 00h-0Eh. Таким образом, сумма всех 16 байт таблицы дополнения FDPT должна быть равна нулю.

Дополнительные дисковые функции

Для обеспечения поддержки новых возможностей интерфейса ATA/ ATAPI (подключение до четырех устройств, новые высокоскоростные режимы передачи данных) в набор функций Int 13h фирмой Phoenix Technologies были введены дополнительные функции (BIOS Extensions).

Дополнительные функции имеют номера 41h-49h и 4Eh. Порядок работы с этими функциями существенно отличается от принятого для стандартных функций прерывания Int 13h.:

- вся адресная информация передается через буфер в оперативной памяти, а не через регистры;
- соглашения об использовании регистров изменены (для обеспечения передачи новых структур данных);
- для определения дополнительных возможностей аппаратуры (параметров) используются флаги.

ПРИМЕЧАНИЕ

Как и «классические» дисковые функции BIOS, дополнительные функции допускают использование режима линейной адресации оперативной памяти.

Пакет дискового адреса

Фундаментальной структурой данных для дополнительных функций прерывания Int 13h является так называемый «Пакет дискового адреса» (Disk Address Packet). Получив пакет дискового адреса, прерывание Int 13h преобразует содержащиеся в нем данные в физические параметры, соответствующие используемому носителю информации.

Формат пакета дискового адреса описан в табл. 6.22.

Таблица 6.22. Формат пакета дискового адреса

Смещение	Размер поля	Назначение поля
00h	BYTE	Размер пакета в байтах
01h	BYTE	Зарезервировано (имеет значение 0)
02h	BYTE	Число передаваемых блоков (0-7Fh) или признак передачи большого массива данных (FFh)
03h	BYTE	Зарезервировано (имеет значение 0)
04h	DWORD	Адрес буфера данных в оперативной памяти в режиме «сегмент:смещение» или признак линейной адресации памяти (FFFF:FFFFh)
08h	QWORD	Абсолютный номер начального блока (LBA-адрес) данных на диске
10h	QWORD	64-разрядный линейный адрес буфера передачи
18h	DWORD	Число передаваемых блоков при передаче большого массива данных
1Ch	DWORD	Зарезервировано (имеет значение 0)

Рассмотрим назначение отдельных полей пакета дискового адреса более подробно.

Байт со смещением 00h содержит размер пакета дискового адреса в байтах. Размер должен составлять 16 и более байт: если значение данного поля меньше 16, то функция завершается аварийно (устанавливается CF = 1, AH = 01h).

Байт со смещением 01h зарезервирован для последующих версий стандарта и должен содержать значение 0.

Байт со смещением 02h содержит число блоков (секторов) данных, подлежащих передаче. Значение числа передаваемых блоков не должно превышать 127 (7Fh), в противном случае функция завершается аварийно (устанавливается CF = 1, AH = 01h).

Если поле содержит значение 0, то при выполнении функции передача данных не производится. Если поле содержит значение FFh, то используется 64-разрядная адресация данных: адрес буфера задается не двойным словом со смещением 04h, а квадрасловом со смещением 10h; число передаваемых блоков задается двойным словом со смещением 18h.

Байт со смещением 03h зарезервирован для последующих версий стандарта и должен содержать значение 0.

Двойное слово со смещением 04h содержит адрес буфера в оперативной памяти, который используется при работе с диском. Адрес должен быть представлен в формате «сегмент/смещение», то есть буфер должен находиться в пределах первого мегабайта адресного пространства процессора. Если данное поле содержит значение FFFF:FFFFh, то для обращения к буферу применяется линейный адрес, который задается квадрасловом со смещением 10h.

Квадраслово со смещением 08h содержит абсолютный 64-разрядный номер начального блока (LBA-адрес) данных на диске. Если устройство поддерживает режим LBA, то данный адрес передается ему непосредственно, без каких-либо модификаций. Если устройство не может работать в режиме LBA, то производится преобразование линейного адреса в формат CHS.

Все перечисленные поля пакета дискового адреса присутствуют в стандарте с момента появления его первой версии. В новых версиях размер пакета был увеличен вдвое (32 байта вместо 16), так как возникла необходимость в поддержке 64-разрядной адресации данных, которая будет применяться в компьютерах с новыми (64-разрядными) моделями процессоров.

Квадраслово со смещением 10h содержит 64-разрядный линейный адрес буфера передачи, то есть буфера в оперативной памяти, который используется при выполнении операций записи и считывания. Данное поле имеет значение только в том случае, если байт со смещением 02h содержит значение FFh или двойное слово со смещением 04h содержит значение FFFF:FFFFh.

Двойное слово со смещением 18h задает число передаваемых блоков, но используется только в том случае, если байт со смещением 02h содержит значение FFh.

Двойное слово со смещением 1Ch зарезервировано для последующих версий стандарта (должно содержать значение 0).

Правила передачи параметров дополнительным функциям

При вызове прерывания дополнительным функциям BIOS передаются через регистры процессора следующие данные:

- в AH — номер вызываемой функции;
- в DL — номер диска;
- в DS:SI — адрес буфера, содержащего пакет дискового адреса.

Передача остальных параметров, как было уже указано выше, производится через пакет дискового адреса.

Дополнительные функции BIOS предназначены только для жестких дисков и дисководов сменных дисков большой емкости, причем функции рассчитаны на использование *не более четырех* устройств. Передаваемый функции номер диска, таким образом, должен находиться в диапазоне 80h-83h.

После выполнения функции в регистре AH выдается код состояния (статус возврата). Кроме принятого для классических функций BIOS стандартного набора кодов возврата, которые перечислены в табл. 6.23, для дополнительных функций введено еще несколько кодов, перечисленных в табл. 6.24.

Таблица 6.23. Стандартные коды состояния (для жестких дисков)

Код	Состояние дисковода
00h	Успешное завершение операции, ошибок нет
01h	Недопустимый номер функции или параметр
02h	Не найден адресный маркер
04h	Сектор не найден
05h	Сброс в исходное состояние не выполнен
07h	Повреждена таблица параметров дисковода
0Ah	Обнаружен дефектный сектор
10h	При чтении по контрольному коду обнаружена неисправимая ошибка

Код	Состояние дисководв
11h	При чтении по контрольному коду была обнаружена и исправлена ошибка
20h	Отказ контроллера
40h	Сбой при выполнении поиска
80h	Диск не отвечает (тайм-аут)
AAh	Дисковод не готов
BBh	Неизвестная ошибка
CCh	Ошибка при записи
E0h	Ошибка регистра состояния

Таблица 6.24. Дополнительные коды состояния дисковода

Код	Состояние дисководв
B0h	Том не заперт
B1h	Том заперт в дисководе
B2h	Том является неперемещаемым
B3h	Том используется
B4h	Счетчик запирания переполнен
B5h	Команда извлечения носителя не выполнена
B6h	Носитель присутствует, но защищен от записи

Подгруппы функций

Дополнительные функции условно разделены на три подгруппы:

- функции для доступа к диску;
- функции для блокировки доступа и смены носителей;
- функции внутреннего назначения.

Функции первой группы применяются при работе с дисками всех типов, функции второй группы — при работе со сменными носителями информации, функции третьей группы используются для внутренних нужд BIOS.

В группу функций доступа к диску входят:

- функция 41h — проверка наличия поддержки дополнительных функций;
- функция 42h — расширенное чтение;
- функция 43h — расширенная запись;

- функция 44h — верификация секторов;
- функция 47h — расширенный поиск;
- функция 48h — чтение параметров диска.

В группу функций блокировки доступа и смены носителей входят:

- функция 41h — проверка наличия поддержки дополнительных функций;
- функция 45h — блокировка/разблокирование диска;
- функция 46h — извлечение диска;
- функция 48h — чтение параметров диска;
- функция 49h — получение расширенного статуса смены диска.

В группу функций EDD входят:

- функция 41h — проверка наличия поддержки дополнительных функций;
- функция 48h — чтение параметров диска;
- функция 4Eh — установка конфигурации аппаратуры.

Как видно из вышеизложенного, проверка наличия поддержки дополнительных функций является обязательной и входит во все три подгруппы. Если после вызова функции 41h флаг CF имеет значение 1, а регистр AH содержит код 01h, то дополнительные функции не поддерживаются BIOS и не могут использоваться прикладной программой.

Прерывание Int 13h, функция 41h: проверка наличия поддержки дополнительных функций

Функция выполняет проверку наличия в BIOS компьютера поддержки дополнительных дисковых функций прерывания Int 13h.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 41h;
- в BX — значение 55AAh;
- в DL — номер дисковода.

Если после завершения выполнения функции флаг CF сброшен, то в регистрах находится следующая информация:

- в AH — номер версии дополнения;
- в AL — информация, предназначенная для внутренних целей BIOS;

- в BX — значение 55AAh;
- в CX — битовая карта свойств используемого интерфейса.

Возвращаемая в регистре CX битовая карта позволяет определить, какое подмножество дополнительных функций реализовано в BIOS. Назначение разрядов битовой карты следующее (если интерфейс обладает определенным свойством, то соответствующий разряд устанавливается в единицу):

- бит 0 — признак поддержки группы функций доступа к диску;
- бит 1 — признак поддержки операций блокировки и смены носителя;
- бит 2 — признак поддержки группы функций EDD;
- бит 3 — признак поддержки 64-разрядных расширений;
- биты 4–15 зарезервированы (должны быть установлены в 0).

Если после завершения выполнения функции флаг CF установлен, то дополнительные функции в BIOS не реализованы. В этом случае в регистре AH будет возвращен код состояния 01h.

Прерывание Int 13h, функция 42h: расширенное чтение

Функция осуществляет передачу секторов с заданной области диска в буфер памяти.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 42h;
- в DL — номер дисковода;
- в DS:SI — пакет дискового адреса.

После завершения операции функция возвращает в регистре AH состояние дисковой системы. В случае аварийного завершения выполнения функции поле счетчика блоков в пакете дискового адреса содержит число блоков, которые были успешно прочитаны (прочитаны до того, как произошла ошибка).

Прерывание Int 13h, функция 43h: расширенная запись

Функция осуществляет передачу секторов из буфера памяти в заданную область диска. Запись данных проводится в режиме верификации, то есть после записи выполняется проверка секторов.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 43h;
- в AL — код режима записи (0 или 1 — запись без верификации, 2 — запись с верификацией);
- в DL — номер дисковода;
- в DS:SI — пакет дискового адреса.

После завершения операции функция возвращает в регистре AH состояние дисковой системы. В случае аварийного завершения выполнения функции поле счетчика блоков в пакете дискового адреса содержит число блоков, которые были успешно записаны (записаны до того, как произошла ошибка). Причиной аварийного завершения данной функции может быть отсутствие у заданного устройства поддержки команды записи данных с верификацией. Проверить наличие поддержки записи с верификацией можно при помощи функции 48h.

Прерывание Int 13h, функция 44h: верификация секторов

Функция осуществляет проверку секторов на диске по их контрольным суммам (без передачи информации между диском и оперативной памятью).

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 44h;
- в DL — номер дисковода;
- в DS:SI — пакет дискового адреса.

После завершения операции функция возвращает в регистре AH состояние дисковой системы. В случае аварийного завершения выполнения функции поле счетчика блоков в пакете дискового адреса содержит число блоков, которые были успешно верифицированы (проверены до того, как произошла ошибка).

Прерывание Int 13h, функция 45h: блокировка/разблокирование диска

Функция позволяет логически блокировать («запирать») и разблокировать («отпирать») сменный носитель в заданном дисководе; допускается также запирание дисковода, в который не установлен

носитель информации. Данная функция позволяет блокировать и разблокировать доступ к жесткому диску, если он поддерживает подмножество команд блокировки.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 45h;
- в AL — код выполняемой операции (0 — «запереть» том в дисководе, 1 — «отпереть» том, 2 — вернуть статус блокировки);
- в DL — номер дисковода.

После завершения операции функция возвращает в регистре AH состояние дисковой системы. В случае успешного выполнения функции в регистре AL будет размещен код состояния диска (0 — диск не заперт, 1 — диск заперт).

Операции блокировки могут быть вложенными: допускается до 255 «запираний» для каждого диска. Диск не может быть физически отперт до тех пор, пока на каждую команду запирания не подана соответствующая команда отпирания. Если количество произведенных операций «запирания» диска превышает 255, то функция завершается аварийно с выдачей кода состояния B4h («Счетчик запирания переполнен»). «Лишние» команды отпирания вызывают аварийное завершение выполнения функции с кодом состояния B0h («Том не заперт»).

Прерывание Int 13h, функция 46h: извлечь сменный носитель

Функция позволяет извлечь носитель информации из заданного дисковода (функция предназначена только для дисководов со сменными носителями).

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 46h;
- в AL — значение 0;
- в DL — номер дисковода.

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

Если после выполнения функции установлен флаг CF, то функция завершилась аварийно. Возможны следующие основные причины аварийного завершения:

- попытка применения данной функции к жесткому диску приводит к аварийному завершению выполнения операции с кодом B2h («Том является неперемещаемым»);
- попытка извлечь «запертый» том приводит к аварийному завершению с кодом B1h («Том заперт в дисководе»);
- попытка извлечь носитель из пустого дисковода приводит к аварийному завершению с кодом 31h («В дисководе нет носителя»);
- неисправность дисковода может привести к аварийному завершению с кодом B5h («Команда извлечения носителя не выполнена»).

Прерывание Int 13h, функция 47h: расширенный поиск

Функция устанавливает головки дисковода на заданный сектор.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 47h;
- в DL — номер дисковода;
- в DS:SI — пакет дискового адреса.

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

Прерывание Int 13h, функция 48h: получить параметры дисковода

Функция устанавливает головки дисковода на заданный сектор.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 48h;
- в DL — номер дисковода;
- в DS:SI — адрес буфера результата.

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

В буфере, на который указывает пара регистров DS:SI, в случае успешного завершения выполнения функции будет возвращена информация о физических параметрах устройства.

Структура информации в буфере результата описана в табл. 6.25. В ранних версиях стандарта использовался буфер диной до 30 байт.

Блок информации о пути к устройству (длиной 44 байта) добавлен в буфер результата сравнительно недавно и поддерживается только в самых современных версиях BIOS. Рассмотрим назначение полей данной структуры более подробно.

Таблица 6.25. Структура информации в буфере результата

Смещение	Размер поля	Описание поля
00h	WORD	Размер буфера в байтах (не менее 26)
02h	WORD	Информационные флаги
04h	WORD	Физическое число цилиндров
08h	DWORD	Физическое число голоаок
0Ch	DWORD	Физическое число секторов на дорожке
10h	QWORD	Общее число секторов на диске
18h	WORD	Число байтов а секторе
1Ah	DWORD	Указатель на конфигурационные параметры EDD
1Eh	WORD	Сигнатура 0BEDDh — признак наличия информации о пути к устройству
20h	BYTE	Длина блока информации о пути к устройству, включая начальную сигнатуру (поле должно иметь значение 44)
21h	BYTE	Зарезервировано (имеет значение 0)
22h	WORD	Зарезервировано (имеет значение 0)
24h	4 байта	Тип шины, записанный в виде строки в ASCII-коде
28h	8 байтов	Тип интерфейса, записанный в виде строки в ASCII-коде (см. табл. 3.17)
30h	8 байтов	Структура — описатель пути к интерфейсу (см. табл. 3.18)
38h	16 байтов	Структура — описатель пути к устройству (см. табл. 3.19)
48h	BYTE	Зарезервировано (имеет значение 0)
49h	BYTE	Контрольная сумма блока информации о пути к устройству (двоичное дополнение суммы байтоа со смещениями 1Eh–48h)

Слово со смещением 00h содержит размер буфера в байтах. Значение данного слова влияет на результат выполнения функции следующим образом:

- если задан размер менее 26 байт, то функция завершается аварийно;

- если размер буфера больше либо равен 26, но меньше 29, то после выполнения функции размер устанавливается равным 26, а указатель на конфигурационные параметры EDD функция не возвращает, так как для него не выделено место в буфере;
- если размер буфера больше либо равен 30, то после выполнения функции он устанавливается равным 30.

Слово со смещением 02h содержит так называемые информационные флаги (если флаг имеет значение 1, то дисковод обладает соответствующим свойством):

- бит 0 — «прозрачное» управление сообщениями о нарушении границ при использовании DMA;
- бит 1 — значения в словах со смещением 08h и 0Ch (физическое число головок и секторов на дорожке) являются достоверными;
- бит 2 — дисковод является устройством со сменными носителями;
- бит 3 — устройство поддерживает режим записи с верификацией;
- бит 4 — устройство вырабатывает сигнал смены носителя;
- бит 5 — устройство способно выполнять «запирание» носителя;
- бит 6 — геометрия устройства сконфигурирована на максимум, носитель информации в устройство не загружен;
- биты 7–15 зарезервированы и должны иметь значение 0.

Биты 4–6 могут принимать значение 1 только в том случае, если установлен бит 2, то есть если дисковод является устройством со сменными носителями. Бит 6 устанавливается при отсутствии носителя и сбрасывается после его загрузки.

Двойное слово со смещением 04h содержит физическое число цилиндров устройства. Значение этого слова на единицу больше максимального номера цилиндра, так как счет цилиндров ведется с нуля.

Двойное слово со смещением 08h содержит физическое число головок устройства. Значение данного слова на единицу больше максимального номера головки, так как счет головок ведется с нуля.

Двойное слово со смещением 0Ch содержит физическое число секторов на дорожке. Значение данного слова равно максимальному номеру сектора, так как счет секторов ведется с единицы.

Квадраслово со смещением 10h содержит общее число физических секторов на носителе. Значение данного слова на единицу больше максимального абсолютного номера сектора, так как счет абсолютных номеров ведется с нуля.

ВНИМАНИЕ

Поля буфера результата со смещением 1Ah и выше не являются обязательными (могут отсутствовать).

Слово со смещением 18h содержит число байтов в секторе диска.

Двойное слово со смещением 1Ah содержит указатель на конфигурационные параметры EDD. Данное поле присутствует в буфере только в том случае, если установлен признак наличия поддержки функций внутреннего назначения (бит 2 регистра CX при выдаче результата функцией 41h прерывания Int 13h). Значение FFFFh:FFFFh в данном поле означает, что указатель недействителен.

Блок информации о пути к устройству имеет длину 44 байта и используется для локализации устройства. Если при вызове функции в буфер результата помещен блок информации о пути к устройству, то слово со смещением 1Eh должно содержать сигнатуру 0BEDDh.

Байт со смещением 20h содержит размер блока информации о пути к устройству в байтах, который должен иметь значение 44.

Байт со смещением 21h и слово со смещением 22h зарезервированы и должны содержать нули.

Строка из четырех ASCII-символов со смещением 24h описывает тип шины, к которой должно быть подключено устройство. Строка может содержать одну из двух возможных последовательностей байтов:

- для шины PCI — последовательность байтов 50h, 43h,49h, 20h;
- для шины ISA — последовательность байтов 49h, 53h, 41h, 20h.

Строка из восьми ASCII-символов со смещением 28h описывает тип интерфейса устройства. Возможные типы интерфейсов и соответствующие им последовательности байтов перечислены в табл. 6.26.

Таблица 6.26. Байтовые последовательности, соответствующие различным типам интерфейсов

Тип интерфейса	Мнемоническое обозначение	Последоввтельность ASCII-кодов
ATA	ATA	41h, 54h, 41h, 20h, 20h, 20h, 20h, 20h
ATAPI	ATAPI	41h, 54h, 41h, 50h, 49h, 20h, 20h, 20h
SCSI	SCSI	53h, 43h, 53h, 49h, 20h, 20h, 20h, 20h
USB	USB	55h, 53h, 42h, 20h, 20h, 20h, 20h, 20h
IEEE1394	1394	31h, 33h, 39h, 34h, 20h, 20h, 20h, 20h
Fibre Channel	FIBRE	46h, 49h, 42h, 52h, 45h, 20h, 20h, 20h
Intelligent Input/ Output	I_2O	49h, 32h, 4Fh, 20h, 20h, 20h, 20h, 20h

Структура со смещением 30h имеет длину 8 байт и содержит описатель пути к интерфейсу устройства. Формат структуры описателя пути к интерфейсу показан в табл. 6.27.

Таблица 6.27. Формат структуры описателя пути к интерфейсу устройства

Тип шины	Смещение от начале буфера результата	Размер поля	Назначение поля
ISA	30h	WORD	16-разрядный базовый адрес
	32h	WORD	Зарезервировано (имеет значение 0)
	34h	DWORD	Зарезервировано (имеет значение 0)
PCI	30h	BYTE	Номер шины
	31h	BYTE	Номер слота
	32h	BYTE	Номер функции
	33h	BYTE	Номер канала
	34h	WORD	Зарезервировано (имеет значение 0)

Структура со смещением 38h имеет длину 16 байт и содержит описатель пути к устройству, который позволяет BIOS получить доступ к конкретному устройству заданного интерфейса. Формат описателя пути к устройству показан в табл. 6.28 (как видно из таблицы, он зависит от типа используемого интерфейса).

Таблица 6.28. Формат структуры описателя пути к интерфейсу устройства

Тип интерфейса	Смещение от начала буфера результата	Размер поля	Назначение поля
ATA	38h	BYTE	Номер устройства на канале (0 — ведущее устройство, 1 — ведомое устройство)
	39h	BYTE	Зарезервировано (имеет значение 0)
	3Ah	WORD	Зарезервировано (имеет значение 0)
	3Ch	DWORD	Зарезервировано (имеет значение 0)
	40h	QWORD	Зарезервировано (имеет значение 0)

Тип интерфейса	Смещение от начала буфера результата	Размер поля	Назначение поля
ATAPI	38h	BYTE	Номер устройства на канале (0 — ведущее устройство, 1 — ведомое устройство)
	39h	BYTE	Номер логического устройства
	3Ah	BYTE	Зарезервировано (имеет значение 0)
	3Bh	BYTE	Зарезервировано (имеет значение 0)
	3Ch	DWORD	Зарезервировано (имеет значение 0)
	40h	QWORD	Зарезервировано (имеет значение 0)
SCSI	38h	WORD	Физический номер устройства (SCSI ID)
	3Ah	QWORD	Номер логического устройства
	42h	WORD	Зарезервировано (имеет значение 0)
	44h	DWORD	Зарезервировано (имеет значение 0)
USB	38h	QWORD	64-разрядный порядковый номер устройства
	40h	QWORD	Зарезервировано (имеет значение 0)
IEEE1394	38h	QWORD	64-разрядный расширенный уникальный идентификатор (EUI-64)
	40h	QWORD	Зарезервировано (имеет значение 0)
Fibre Cannel	38h	QWORD	64-разрядное слово идентификатора (WWID)
	40h	QWORD	Номер логического устройства
I_2O	38h	QWORD	64-разрядный идентификационный тег
	40h	QWORD	Зарезервировано (имеет значение 0)

ПРИМЕЧАНИЕ

Поле «Номер канала» структуры описателя пути для шины PCI необходимо для того, чтобы можно было различать интерфейсы, имеющие идентичные координаты на шине PCI, то есть одинаковые номера шины, слота и функции. Например, интерфейс ATA допускает наличие на системной плате первичного (Primary) и вторичного (Secondary) каналов, у которых номера шины, слота и функции совпадают. В этом случае Primary-каналу присваиваается номер 0, а Secondary-каналу — номер 1.

Байт со смещением 48h зарезервирован и должен содержать значение 0.

Байт со смещением 49h содержит контрольную сумму байтов блока информации о пути к устройству (начиная с сигнатуры блока). Контрольная сумма вычисляется как двоичное дополнение суммы байтов со смещениями 1Eh-48h; в результате сумма байтов 1Eh-49h должна быть равно нулю.

Прерывание Int 13h, функция 49h: получение расширенного статуса смены диска

Функция возвращает статус смены диска. Она аналогична функции 16h прерывания Int 13h, но допускает задание любого номера дисковода.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 49h;
- в DL — номер дисковода.

После завершения операции функция возвращает в регистре AH состояние дисковой системы. Если флаг CF сброшен и AH = 0, то сигнал смены диска неактивен; если флаг CF установлен и AH = 06h, то сигнал смены диска активен.

Если в качестве параметра данной функции указан номер жесткого диска, поддерживающего подгруппу команд запирания и извлечения носителя, то функция всегда завершается успешно: после ее выполнения флаг CF сброшен и AH = 0.

ПРИМЕЧАНИЕ

Сигнал смены диска активируется при открывании и закрывании дверцы накопителя независимо от того, производилась ли при этом замена носителя.

Прерывание Int 13h, функция 4Eh: установка конфигурации аппаратуры

Функция позволяет аппаратно-независимому программному обеспечению настраивать устройства системной платы на оптимальный режим работы.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 4Eh;
- в AL — номер подфункции конфигурирования аппаратуры (табл. 6.29);
- в DL — номер дисковода.

Таблица 6.29. Список подфункций конфигурирования аппаратуры

Номер подфункции	Назначение
0	Разрешить предвыборку
1	Запретить предвыборку
2	Установить максимально допустимый номер режима PIO и повысить скорость передачи данных до максимально допустимой для данного диска и системной платы
3	Установить режим PIO 0 и понизить до минимума скорость передачи данных
4	Установить режим PIO, используемый по умолчанию. Вернуть систему к скорости передачи данных, заданной процедурой SETUP
5	Установить максимально допустимый для прерывания Int 13h номер режима DMA
6	Запретить прерыванию Int 13h использование DMA

После завершения операции функция возвращает в регистре AH состояние дисковой системы.

В случае успешного выполнения функции в регистре AL возвращается информация о возможном воздействии на другое устройство, подключенное к тому же каналу ATA (0 — функция выполнена корректно, 1 — другое устройство, подключенное к данному каналу, подверглось воздействию). Дело в том, что на некоторых системных платах до сих пор встречаются устаревшие чипсеты, которые не обеспечивают взаимной независимости режимов работы устройств, подключенных к одному ATA-каналу.

ВНИМАНИЕ

Режимы PIO и DMA не могут использоваться одновременно. Установка режима DMA отменяет режим PIO и сбрасывает заданные для него параметры скорости передачи данных. Аналогично установка режима PIO отменяет режим DMA и сбрасывает заданные для него параметры.

Прерывание Int 15h, функция 52h: извлечь сменный носитель

Функция позволяет извлечь из дисковода сменный носитель; она считается «низкоуровневой» — служит буфером между устройством и функцией 46h прерывания Int 13h. Данная функция перехватывается программами кэширования данных на диске, что позволяет предотвратить потерю информации.

Перед вызовом прерывания требуется записать в регистры следующие значения:

- в AH — значение 52h;
- в DL — номер дисковода.

После завершения операции функция возвращает в регистре AH состояние дисковой системы. Если флаг CF сброшен и AH = 0, то заданная операция успешно выполнена, а если флаг CF установлен, то произошла ошибка, и ее код помещен в регистр AH.

Файловые системы FAT12, FAT16 и FAT32

Между заголовками разных уровней нужно вставить текст, относящийся ко всему заголовку второго уровня, хотя бы одно предложение.

Форматы адресации данных LBA и CHS

Если у программиста возникает потребность в работе с диском на низком уровне (то есть на уровне BIOS или дискового контроллера), то он должен знать формат носителя информации, чтобы получить доступ к данным. Понятие «формат носителя информации» включает в себя:

- структуру информации на носителе (физический формат);
- способы адресации элементов физической структуры (логический формат).

При работе с жесткими дисками программисты всегда оперируют логическим форматом носителя, поскольку физический формат является скрытым и доступен только встроенному микропроцессору диска. Иметь дело с физическим форматом приходится только в случае работы с гибкими дисками на низком уровне, то есть на уровне контроллера дисковода.

Как уже было упомянуто выше, физический адрес сектора на диске может быть задан или в формате линейного адреса (LBA), или в формате «цилиндр-головка-сектор» (CHS). Линейная адресация применяется для жестких дисков и сменных носителей большой емкости, а формат CHS — в основном для гибких дисков.

Режим LBA отличается исключительно простотой: для обращения к информации на диске используется только один параметр — абсолютный (логический) адрес сектора. Счет секторов ведется с нуля, а максимальный номер сектора равен $2^{28}-1$.

Формат CHS неудобен тем, что программист вынужден выполнять дополнительные преобразования адреса сектора данных, к которому производится обращение — пересчитывать линейный (логический) адрес в трехмерную систему координат, включающую номер цилиндра (дорожки) C, номер поверхности диска (головки) H и номер сектора на дорожке S. При этом программист должен учитывать, что счет номеров цилиндров и поверхностей ведется с нуля, а счет номеров секторов — с единицы. Если в документации дисковода указано, что он имеет C_{max} цилиндров, H_{max} поверхностей и S_{max} секторов на дорожке, то значение C может изменяться от 0 до (C_{max}-1), значение H — от 0 до (H_{max}-1), а значение S — от 1 до S_{max}.

Соответствие между физическим адресом сектора в формате CHS и его логическим номером N определяется следующей формулой:

$$N = (C \times H_{max} + H) \times S_{max} + S - 1$$

Обратите внимание, что счет логических номеров ведется с нуля. Для обратного преобразования N в CHS используется целочисленное деление (остаток отбрасывается):

$$C = N / (H_{max} \times S_{max})$$
$$H = (N - C \times H_{max} \times S_{max}) / S_{max}$$
$$S = N - (C \times H_{max} + H) \times S_{max} + 1$$

Предельные значения величин C_{max}, H_{max} и S_{max} определяются количеством двоичных разрядов, выделенных для их хранения в регистрах контроллера диска. Для жестких дисков они могут достигать значений 65 536, 16 и 255 соответственно (при использовании стандартных функций BIOS возникают дополнительные ограничения: предельное число цилиндров — 1024, секторов — 63).

Максимально возможный объем диска ATA-типа равен произведению числа секторов на диске на размер сектора в байтах — 128 Гбайт в режиме LBA и 127,5 Гбайт в режиме CHS (то есть почти не зависит от режима адресации). В настоящий момент указанный предел практически достигнут — уже выпускаются диски объемом 80 Гбайт. Дисководы с интерфейсом SCSI изначально рассчитаны только на линейную адресацию данных и позволяют адресовать до 2^{32} секторов, то есть могут хранить до 2 Тбайт информации.

Размещение информации на логических дисках

Операционная система выбирает способ организации хранения информации на носителе в зависимости от его типа и объема, а также пожеланий пользователя. Гибкие диски для АТ-совместимых компьютеров всегда организованы в виде одного логического диска со структурой FAT12. Жесткий диск может содержать один или несколько разделов, предназначенных для одной или нескольких различных операционных систем, а разделы в свою очередь могут состоять из одного или нескольких логических дисков.

Логический диск (том) файловой системы типа FAT состоит из четырех основных областей (рис. 6.1), расположенных в следующем порядке:

- резервная область;
- область таблиц размещения файлов (FAT1 и FAT2);
- область корневого каталога (не существует в FAT32);
- область файлов и каталогов.



Рис. 6.1. Организация данных на логических дисках

В первом секторе логического диска с системой FAT располагаются загрузочный сектор и блок параметров BIOS. В документации Microsoft они обозначаются как Boot Sector (BS) и BIOS Parameter Block (BPB) соответственно. Начальный участок данного блока для всех типов FAT идентичен; описание этого участка приведено в табл. 6.30.

Таблица 6.30. Начальный участок загрузочного сектора

Наименование элемента	Смещение	Размер, байт	Описание
BS_jmpBoot	00h	3	Инструкция перехода (jmp) на загрузочный код
BS_OEMName	03h	8	Текстовая строка с аббревиатурой фирмы-изготовителя и номером версии операционной системы
BPB_BytsPerSec	0Bh	2	Число байтов в секторе (всегда 512)
BPB_SecPerClus	0Dh	1	Число секторов в кластере
BPB_RsvdSecCnt	0Eh	2	Число резервных секторов в резервной области раздела, начиная с первого сектора раздела
BPB_NumFATs	10h	1	Число таблиц (копий) FAT в разделе (всегда равно 2)
BPB_RootEntCnt	11h	2	Для FAT12 и FAT16 — количество 32-байтных дескрипторов файлов в корневом каталоге (при использовании FAT16 равно 512); для FAT32 это поле имеет значение 0
BPB_TotSec16	13h	2	Общее число секторов в разделе (если данное поле имеет значение 0, то число секторов задается полем BPB_TotSec32)
BPB_Media	15h	1	Тип носителя информации (см. табл. 6.33)
BPB_FATSz16	16h	2	Для FAT12 и FAT16 — количество секторов, занимаемых одной копией FAT; для FAT32 поле имеет значение 0
BPB_SecPerTrk	18h	2	Число секторов на дорожке (для прерывания 13h)
BPB_NumHeads	1Ah	2	Число головок (для прерывания 13h)
BPB_HiddSec	1Ch	4	Число скрытых секторов, предшествующих разделу, содержащему данный том. Значение равно 0 для носителей, не подлежащих разбиению на разделы

продолжение ⇒

Таблица 6.30 (*продолжение*)

Наименование элемента	Смещение	Размер, байт	Описание
BPB_TotSec32	20h	4	Общее число секторов в разделе (поле используется вместо BPB_TotSec16, если в разделе свыше 65 535 секторов; в противном случае поле содержит значение 0)

Информацию, приведенную в табл. 6.30, нужно дополнить следующими замечаниями.

- Поле безусловного перехода BS_jmpBoot может иметь два формата, соответствующих двум разным типам инструкций перехода процессора x86. Первый вариант начинается с кода EBh, второй — с кода E9h.
- Поле BS_OEMName, вообще говоря, может содержать любое значение, однако современные операционные системы Microsoft заносят в него код MSWIN4 1.
- Поле BPB_SecPerClus задает число секторов в кластере и может содержать значения, равные 2^N: 1, 2, 4, 8, 16, 32, 64.
- Поле BPB_RsvdSecCnt задает количество секторов резервной области. Для FAT12 и FAT16 данное поле всегда имеет значение 1, а для FAT32 — обычно значение 32.
- Поля BPB_TotSec16 и BPB_TotSec32 задают число секторов в разделе, причем это число заносится только в одно из указанных полей, а второе должно содержать значение 0. Если в разделе не более 65 535 секторов, то используется поле BPB_TotSec16, иначе — поле BPB_TotSec32.
- Поле BPB_Media описывает тип установленного в дисковод носителя информации. Перечень допустимых кодов носителей приведен в табл. 6.33.

Различия в структуре загрузочных секторов для разных типов FAT начинаются со смещения 24h. Для FAT12 и FAT16 структура имеет вид, показанный в табл. 6.31, а для FAT32 — в табл. 6.32. Рассмотрим отдельные поля более подробно.

- Поле BS_FilSysType содержит аббревиатуру файловой системы (FAT12, FAT16 или FAT32), но не должно использоваться для идентификации ее типа.

- Бит 7 поля BPB_ExtFlags содержит признак активности FAT: если он равен нулю, то в процессе работы изменения отражаются во всех FAT, а в противном случае активной является только одна копия, номер которой записан в битах 0–3 поля BPB_ExtFlags (счет номеров ведется с 0). Остальные разряды поля BPB_ExtFlags зарезервированы.

Таблица 6.31. Структура загрузочного сектора для FAT12 и FAT16, начиная со смещения 24h

Наименование элемента	Смещение	Размер, байт	Описание
BS_DrvNum	24h	1	Номер дисковода для прерывания 13h (для дисководов гибких дисков — от 0 до 3; для жестких дисков счет номеров ведется с 80h)
BS_Reserved1	25h	1	Зарезервировано для Windows NT, имеет значение 0
BS_BootSig	26h	1	Признак расширенной загрузочной записи (29h). Показывает, что следующие три поля присутствуют
BS_VolID	27h	4	Номер логического диска (формируется при форматировании как комбинация времени и даты создания)
BS_VolLab	2Bh	11	Метка диска (текстовая строка)
BS_FilSysType	36h	8	Текстовая строка с аббревиатурой типа файловой системы

Таблица 6.32. Структура загрузочного сектора для FAT32, начиная со смещения 24h

Наименование элемента	Смещение	Размер, байт	Описание
BPB_FATSz32	24h	4	Количество секторов, занимаемых одной копией FAT
BPB_ExtFlags	28h	2	Номер активной FAT
BPB_FSVer	2Ah	2	Номер версии FAT32: старший байт — номер версии, младший — номер ревизии. В настоящее время используется значение 0:0

продолжение ⇨

Таблица 6.32 (*продолжение*)

Наименование элемента	Смещение	Размер, байт	Описание
BPB_RootClus	2Ch	4	Номер кластера для первого кластера корневого каталога (обычно имеет значение 2)
BPB_FSInfo	30h	2	Номер сектора структуры FSINFO в резервной области логического диска (обычно 1)
BPB_BkBootSec	32h	2	Номер сектора (в резервной области логического диска), используемого для хранения резервной копии загрузочного сектора (обычно 6)
BPB_Reserved	34h	12	Область, зарезервированная для дальнейших расширений (должна содержать нули)
BS_DrvNum	40h	1	Номер дисковода для прерывания 13h (для дисководов гибких дисков — от 0 до 3; для жестких дисков счет номеров ведется с 80h)
BS_Reserved1	41h	1	Зарезервировано для Windows NT, имеет значение 0
BS_BootSig	42h	1	Признак расширенной загрузочной записи (29h). Показывает, что следующие три поля присутствуют
BS_VolID	43h	4	Номер логического диска (формируется при форматировании как комбинация времени и даты создания)
BS_VolLab	47h	11	Метка диска (текстовая строка)
BS_FilSysType	52h	8	Текстовая строка с аббревиатурой типа файловой системы

Таблица 6.33. Типы носителей информации

Код типа	Тип носителя информации
F0h	Гибкий диск, 2 стороны, 18 секторов на дорожке
F8h	Жесткий диск
F9h	Гибкий диск, 2 стороны, 15 секторов на дорожке
FCh	Гибкий диск, 1 сторона, 9 секторов на дорожке

Код типа	Тип носителя информации
FDh	Гибкий диск, 2 стороны, 9 секторов на дорожке
Feh	Гибкий диск, 1 сторона, 8 секторов на дорожке
FFh	Гибкий диск, 2 стороны, 8 секторов на дорожке

Кроме перечисленных в табл. 6.30–6.32 полей, нулевой сектор логического диска должен содержать в байте со смещением 1FEh код 55h, а в следующем байте (со смещением 1FFh) — код AAh. Указанные два байта являются сигнатурой — признаком загрузочного сектора логического диска.

Таким образом, загрузочный сектор выполняет две важные функции: описывает структуру данных на диске, а также позволяет осуществить загрузку операционной системы (если она установлена на диске, и диск является активным). В случае возникновения ошибки в момент записи информации в загрузочный сектор (например, из-за внезапного выключения электропитания) может быть утрачена вся информация на логическом диске, поэтому в системе FAT32 сектор 6 резервной области используется для хранения копии загрузочного сектора (Backup Boot Sector).

Поскольку размер таблицы FAT на логическом диске с организацией FAT32 может быть очень велик, для ускорения выполнения операций с FAT была введена структура FSInfo, размещаемая в секторе 1 резервной области. Эта структура содержит информацию о количестве свободных кластеров на диске и о номере первого свободного кластера в таблице FAT. Формат структуры описан в табл. 6.34.

Таблица 6.34. Структура сектора FSInfo и резервного загрузочного сектора FAT32

Наименование элемента	Смещение	Размер, байт	Описание
FSI_LeadSig	000h	4	Значение 41615252h — сигнатура, которая служит признаком того, что данный сектор содержит структуру FSInfo
FSI_Reserved1	004h	480	Поле зарезервировано для дальнейших расширений (все байты должны быть заполнены нулями)
FSI_StrucSig	1E4h	4	Значение 61417272h (сигнатура)

продолжение ⇗

Таблица 6.34 (*продолжение*)

Наименование элемента	Смещение	Размер,	Описание
FSI_Free_Count	1E8h	4	Содержит текущее число свободных кластеров на диске. Если в поле записано значение 0FFFFFFFFh, то количество свободных кластеров неизвестно (его нужно вычислять)
FSI_Nxt_Free	1ECh	4	Содержит номер кластера, с которого дисковый драйвер должен начинать поиск свободных кластеров. Если в поле записано значение 0FFFFFFFFh, то поиск свободных кластеров нужно начинать с кластера № 2
FSI_ Reserved2	1F0h	12	Поле зарезервировано для дальнейших расширений (все байты должны быть заполнены нулями)
FSI_TrailSig	1FCh	4	Значение AA550000h — сигнатура, которая служит признаком конца структуры FSInfo

Хотя относительные номера секторов структур Backup Boot Sector и FSInfo являются константами, значения этих констант зачем-то хранятся в полях BPB_BkBootSec и BPB_FSInfo загрузочного сектора.

Чтобы прочесть значение некоторого элемента любой структуры данных, нужно вначале получить доступ к самой структуре (то есть определить ее местоположение на диске), а затем прочитать с диска в буфер оперативной памяти либо всю структуру, либо только тот сектор, в котором находится интересующий нас элемент. Местоположение основных структур данных на физическом диске, то есть абсолютные номера начальных секторов, можно вычислить по следующим формулам:

```
BS_StartSect = LDisk_StartSect
FSInfo_StartSect = BS_StartSect + BPB_FSInfo
FAT1_StartSect  = BS_StartSect + BPB_RsvdSecCnt
FAT2_StartSect = FAT1_StartSect + BPB_FATSz
RDir_StartSect = FAT1_StartSect + BPB_FATSz × BPB_NumFATs
Data_StartSect = RDir_StartSect + (32 × BPB_RootEntCnt)/512
```

В приведенных формулах выделены жирным шрифтом переменные, значения которых необходимо извлечь из структур данных, расположенных в загрузочном секторе логического диска, и использованы следующие обозначения:

- BS_StartSect — номер загрузочного сектора логического диска;
- LDisk_StartSect — номер начального сектора диска;
- FAT1_StartSect — номер начального сектора первой FAT;
- BPB_RsvdSecCnt — число секторов в резервной области (32 для системы FAT32 и 1 для остальных систем);
- FAT2_StartSect — номер начального сектора второй FAT;
- BPB_FATSz — размер одной копии FAT (BPB_FATSz32 для FAT32 и BPB_FATSz16 для остальных систем);
- BPB_NumFATs — число копий FAT на диске;
- RDir_StartSect — номер начального сектора корневого каталога;
- Data_StartSect — номер начального сектора области данных;
- BPB_RootEntCnt — число записей в корневом каталоге (0 для FAT32 и 512 для остальных систем).

В листинге 6.6 приведена программа ShowFDDSector, позволяющая при помощи функций BIOS осуществлять последовательный просмотр секторов гибкого диска в ASCII-кодах. Перед тем как начать просмотр, программа просит пользователя установить гибкий диск в дисковод A:, считывает загрузочный сектор и извлекает из него информацию о формате гибкого диска (то есть о количестве рабочих поверхностей диска и числе секторов на дорожке). Пролистывание секторов осуществляется при помощи клавиш ↓ и ↑; для выхода из программы нужно нажать клавишу Esc. Для считывания секторов используется вспомогательная процедура ReadFDDSector.

Листинг 6.6. Просмотр секторов гибкого диска на дисководе A: при помощи функций BIOS

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"
```

продолжение

Листинг 6.6 (*продолжение*)

```
DATASEG
; Текстовые сообщения
Txt0 DB LIGHTCYAN,0,7,"ПРИМЕР ИСПОЛЬЗОВАНИЯ ФУНКЦИЙ "
     DB "BIOS ДЛЯ ЧТЕНИЯ ИНФОРМАЦИИ С ДИСКЕТЫ",0
     DB LIGHTGREEN,12,24
     DB "Установите дискету в дисковод A:",0
     DB YELLOW,24,29,"Нажмите любую клавишу",0
Txt1 DB LIGHTCYAN,0,10,"Просмотр секторов "
     DB "гибкого диска, находящегося в дисководе A:",0
     DB LIGHTGREEN,2,14,"Диск имеет   поверхность(и),"
     DB "    секторов на дорожке",0
     DB LIGHTCYAN,17,8,"Управляющие клавиши:",0
     DB YELLOW,24,27,"Нажмите управляющую клавишу",0
Txt2 DB 19,8,"Стрелка вниз - следующий сектор:",0
     DB 20,8,"Стрелка вверх - предыдущий сектор:",0
     DB 21,8,"Esc - выход.",0
Txt3 DB LIGHTGREEN,5,8
     DB "Дорожка N    Головка N    Сектор N     ",0
TErr DB 12,27,"Ошибка считывания сектора",0

; Адрес считываемого сектора в режиме CHS
Cylinder    DB ?  ;номер цилиндра
Head        DB ?  ;номер головки
Sector      DB ?  ;номер сектора
; Предельные значения координат
MaxCylinder DB 79 ;?  ;максимальный номер цилиндра
MaxHead     DB 1  ;?  ;максимальный номер головки
MaxSector   DB 18 ;?  ;максимальный номер сектора

; Область памяти для хранения прочитанного сектора
SectorDataBuffer DB 512 DUP (?)
ENDS

SEGMENT sseg para stack 'STACK'
        DB 400h DUP(?)
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC ShowFDDSector
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
```

```
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести на экран сообщение "Вставьте дискету"
        MShowColorText 3,Txt0
        ; Ожидать нажатия клавиши
        call    GetChar
; Прочитать BOOT-сектор дискеты
        mov     [Cylinder],0
        mov     [Head],0
        mov     [Sector],1
        call    ReadFDDSector
; Определить и запомнить параметры дискеты
        mov     AL,[SectorDataBuffer+18h]
        mov     [MaxSector],AL
        mov     AL,[SectorDataBuffer+1Ah]
        dec     AL
        mov     [MaxHead],AL

; Очистить экран
        call    ClearScreen
; Вывести текстовые сообщения
        MShowColorText 4,Txt1
; Установить зеленый цвет и черный фон
        mov     [TextColorAndBackground],LIGHTGREEN
; Отобразить параметры установленной дискеты
        mov     AH,[MaxHead]
        inc     AH
        MShowDecByte 2,25,AH
        MShowDecByte 2,43,[MaxSector]
; Вывести описание управляющих клавиш
        MShowText 3,Txt2

; ЦИКЛ ПО СЕКТОРАМ
        mov     AX,0B800h
        mov     ES,AX
@@ReadSector:
        MShowColorString Txt3
        MShowDecByte  5,18,[Cylinder]
        MShowDecByte  5,31,[Head]
        MShowDecByte  5,44,[Sector]
; Прочитать сектор
        call    ReadFDDSector
; Отобразить на экран содержимое считанного сектора
; в ASCII-кодах
@@ShowSector:
        ; Установить начало окна отображения сектора
        mov     DI,7*160+8*2
        mov     SI,offset SectorDataBuffer
        ; Задать для символов светло-голубой
```

продолжение

Листинг 6.6 (*продолжение*)

```
        ; цвет и синий фон
        mov     AH,LIGHTCYAN+BLUE*16
        mov     DX,8  ;счетчик строк
@@OutNextString:
        mov     CX,64 ;счетчик символов в строке
@@OutNextChar:
        lodsb
        stosw
        loop    @@OutNextChar
        add     DI,16*2
        dec     DX
        jnz     @@OutNextString

@@GetCommand:
        ; Ожидаем ввода следующей команды
        call    GetChar
        cmp     AL,0
        je      @@TestCommandByte
        call    Beep
        jmp short @@GetCommand

@@TestCommandByte:
        cmp     AH,B_Esc     ;команда "Выход"?
; Выполнить команду "Выход"
        je      @@End
@@TestDn:
        cmp     AH,B_DN
        jne     @@TestUp
; Выполнить команду "Стрелка вниз"
        mov     AL,[Sector]
        cmp     AL,[MaxSector]
        jae     @@IncHead
        ; Увеличить на 1 номер сектора
        inc     [Sector]
        jmp     @@ReadSector
@@IncHead:
        mov     AL,[Head]
        cmp     AL,[MaxHead]
        jae     @@IncCylinder
        ; Увеличить на 1 номер головки,
        ; установить сектор 1
        inc     [Head]
        mov     [Sector],1
        jmp     @@ReadSector
@@IncCylinder:
        mov     AL,[Cylinder]
        cmp     AL,[MaxCylinder]
```

```
        jae     @@CommandError
        ; Увеличить на 1 номер цилиндра, установить
        ; головку 0, сектор 1
        inc     [Cylinder]
        mov     [Head],0
        mov     [Sector],1
        jmp     @@ReadSector
@@TestUp:
        cmp     AH,B_UP
        jne     @@CommandError
; Выполнить команду "Стрелка вверх"
        cmp     [Sector],1
        jbe     @@DecHead
        ; Уменьшить на 1 номер сектора
        dec     [Sector]
        jmp     @@ReadSector
@@DecHead:
        mov     AL,[Head]
        cmp     AL,0
        je      @@DecCylinder
        ; Уменьшить на 1 номер головки, установить
        ; максимальный номер сектора
        dec     [Head]
        mov     AL,[MaxSector]
        mov     [Sector],AL
        jmp     @@ReadSector
@@DecCylinder:
        mov     AL,[Cylinder]
        cmp     AL,0
        je      @@CommandError
        ; Уменьшить на 1 номер цилиндра, установить
        ; максимальные номера головки и сектора
        dec     [Cylinder]
        mov     AL,[MaxHead]
        mov     [Head],AL
        mov     AL,[MaxSector]
        mov     [Sector],AL
        jmp     @@ReadSector
@@CommandError:
        call    Beep
        jmp     @@GetCommand

@@End:  ; Переустановить текстовый режим
        mov     ax,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP ShowFDDSector
```

продолжение

Листинг 6.6 (*продолжение*)

```
;*********************************************
;*       ПРОЧИТАТЬ СЕКТОР ГИБКОГО ДИСКА       *
;* Параметры передаются через глобальные      *
;* переменные:                                *
;* Cylinder - номер цилиндра (от 0 до 79);    *
;* Head - номер головки (0 или 1);            *
;* Sector - номер сектора (от 1 до 18).       *
;* Считанный сектор записывается в основной   *
;* сегмент данных по адресу SectorDataBuffer. *
;*********************************************
PROC ReadFDDSector NEAR
        pushad
        push    ES
        mov     AX,DS
        mov     ES,AX
        mov     SI,3    ;счетчик повторений
@@Repeat:
        mov     BX,offset SectorDataBuffer
        mov     AH,2    ;подфункция "Прочесть сектор"
        mov     AL,1    ;прочесть 1 сектор
        mov     CH,[Cylinder]
        mov     CL,[Sector]
        mov     DH,[Head]
        mov     DL,0    ;читать диск "A:"
        int     13h
        jnc     @@End
        ; Ошибка считывания, инициализировать систему
        mov     AH,0    ;подфункция "Инициализировать"
        mov     DL,0    ;диск "A:"
        int     13h
        dec     SI
        jnz     @@Repeat
; Аварийный выход - ошибка при чтении файла
        MFatalError TErr
; Нормальное завершение процедуры
@@End:  pop     ES
        popad
        ret
ENDP ReadFDDSector
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуры перевода чисел в десятичный код
include "list2_05.inc"

END
```

ПРИМЕЧАНИЕ

Приведенный пример универсален, и его можно запускать на любом персональном компьютере — даже на древних машинах типа XT с пятидюймовыми дисководами.

Назначение и внутренняя организация таблиц размещения файлов

Размер файла, вообще говоря, не является постоянной величиной и может изменяться (обычно – увеличиваться) со временем. Если допускается хранение файла только в смежных (последовательно расположенных) секторах, то при увеличении размера файла операционная система должна полностью перезаписывать его в другую (свободную) область диска подходящего объема. Чтобы упростить и ускорить выполнение операции добавления новых данных в файл, в современных операционных системах применяются таблицы размещения файлов (File Allocation Table, сокращенно FAT), позволяющие хранить файл в виде нескольких несмежных участков.

При использовании FAT область данных логического диска разделена на участки одинакового размера – кластеры. Кластер может состоять из одного или нескольких последовательно расположенных на диске секторов. Число секторов в кластере должно быть кратно 2^N и может принимать значения от 1 до 64 (размер кластера зависит от типа используемой системы FAT и объема логического диска).

Каждому кластеру поставлен в соответствие собственный элемент таблицы FAT. Первые два элемента FAT являются резервными – если на диске имеется CountOfClusters кластеров данных, то число элементов таблицы будет равно CountOfClusters+2. Тип FAT определяется значением CountOfClusters:

- если CountOfClusters < 4085, то используется система FAT12;
- если 4084 < CountOfClusters < 65525, то используется система FAT16;
- если 65524 < CountOfClusters – используется система FAT32.

Названия типов FAT ведут свое происхождение от размера элемента: злемент FAT12 имеет размер 12 бит (1,5 байта), FAT16 – 16 бит (2 байта), FAT32 – 32 бита (4 байта). Следует учитывать, что в FAT32 четыре старших двоичных разряда зарезервированы и игнорируются в процессе работы операционной системы (то есть

значащими являются только семь младших шестнадцатеричных разрядов элемента).

Каждому файлу, находящемуся в области данных диска, соответствует цепочка элементов FAT — упорядоченный однонаправленный список (рис. 6.2). Поскольку список однонаправленный, он обеспечивает только движение вперед — если вы захотите вернуться к предыдущему кластеру, вам придется снова проводить поиск с самого начала списка.

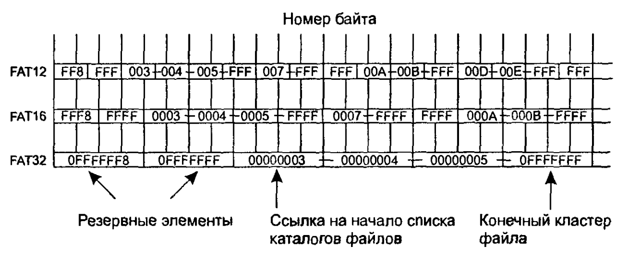


Рис. 6.2. Вид начальных фрагментов для FAT различных типов

В каталоге файлов для каждого файла приводится номер начального элемента в таблице FAT, соответствующий первому кластеру файла. Если файл содержит более одного кластера, то указанный элемент содержит номер элемента FAT, соответствующего следующему кластеру файла. Последний элемент списка содержит признак конца файла (End Of Clasterchain, сокращенно EOC), который в FAT12 может принимать значения от FF8h до FFFh, в FAT16 — значения FFF8h–FFFFh, в FAT — значения FFFFFF8h–FFFFFFFh (табл. 6.35). Операционные системы Microsoft всегда применяют для EOC значения FFFh, FFFFh и FFFFFFFh соответственно, однако дисковые утилиты других фирм могут использовать и иные допустимые значения.

Таблица 6.35. Значения специальных кодов элементов FAT

Значение кода	FAT12	FAT16	FAT32
Свободный кластер	0	0	0
Дефектный кластер	FF7h	FFF7h	FFFFFF7h
Последний кластер в списке	FF8h–FFFh	FFF8h–FFFFh	FFFFFF8h–FFFFFFFh

Кроме признака EOC, определены также специальные коды для свободного кластера (имеет значение 0 во всех типах FAT) и дефектного кластера (имеет значение FF7h в FAT12, значение FFF7h в FAT16 и значение FFFFFF7h в FAT32).

В документации Microsoft для элементов FAT используется принятое в языке программирования С обозначение элементов массивов. Значение, хранящееся в элементе FAT[0] (первом резервном элементе таблицы FAT) является сигнатурой. Для FAT12 оно всегда равно FF8h, для FAT16 — FFF8h, для FAT32 — FFFFFF8h.

В FAT[1] (то есть во второй резервный элемент) при форматировании диска записывается код EOC. Кроме того, системы FAT16 и FAT32 могут использовать два старших значащих разряда указанного элемента в качестве флагов. Флаг ClnShutBitMask занимает в системе FAT16 двоичный разряд 15, а в системе FAT32 разряд 27. Если флаг ClnShutBitMask установлен в 1, то логический диск (том) является «чистым» (clean), если сброшен в 0 — «грязным» (dirty). Термин «грязный» означает, что работа с диском не была завершена надлежащим образом (например, по причине внезапного отключения электропитания) и при загрузке операционной системы должна быть выполнена процедура восстановления диска.

Флаг HrdErrBitMask служит признаком наличия сбоев при выполнении операций ввода-вывода. В системе FAT16 он занимает двоичный разряд 14, а в системе FAT32 — разряд 26. При загрузке операционной системы (точнее, в момент монтирования тома) HrdErrBitMask устанавливается в 1, но в случае возникновения сбоя при записи или считывании информации флаг сбрасывается в 0.

Выполнять операции с элементами таблиц типов FAT16 и FAT32 очень легко, поскольку эти таблицы представляют собой массивы 16-разрядных и 32-разрядных слов соответственно. Работать с элементами FAT12 менее удобно, так как для доступа к элементу массива приходится выполнять ряд вспомогательных действий. Порядок действий при извлечении элемента из FAT12 описан ниже.

1. Умножить номер элемента на 3.
2. Разделить результат на 2.
3. Извлечь из FAT 16-разрядное слово, используя в качестве адреса результат предыдущей операции.
4. Если номер элемента четный, выполнить операцию AND над считанным словом и маской 0FFFh. Если номер нечетный, сдвинуть считанное слово вправо на 4 разряда. В результате получаем искомое значение элемента FAT.

Теперь рассмотрим порядок действий при записи элемента в FAT12.

1. Умножить номер элемента на 3.
2. Разделить результат на 2.
3. Извлечь из FAT 16-разрядное слово, используя в качестве адреса результат предыдущей операции (адрес слова запомнить).
4. Если номер элемента четный, выполнить операцию AND над считанным словом и маской 0F000h, а затем операцию OR над полученным результатом и значением записываемого элемента. Если номер нечетный, выполнить операцию AND над считанным словом и маской 0F000h, а затем сдвинуть значение элемента влево на 4 разряда и выполнить OR с результатом предыдущей операции.
5. Записать полученное 16-разрядное слово обратно в FAT.

В программах, написанных на ассемблере, для выполнения умножения на 3 вместо команды MUL часто применяется алгоритм «сдвиг и сложение»: исходное число копируется, над копией числа выполняется сдвиг влево на один разряд (умножение на 2), а затем оба числа складываются ($x + 2x = 3x$). Вместо команды DIV при делении на 2 используется сдвиг вправо на один разряд.

Элемент FAT содержит номер кластера, но при работе с дисками на низком уровне адресуемой единицей данных является сектор, а не кластер. Номер начального сектора кластера SectorNum связан с номером кластера ClusterNum следующей формулой:

SectorNum = Data_StartSect + (ClusterNum-2)×BPB_SecPerClus

В документации Microsoft приводятся также следующие примечания относительно FAT.

- Данные в последнем секторе FAT не обязательно полностью принадлежат FAT — конец сектора, не относящийся к FAT, может содержать произвольные данные. Последний элемент FAT имеет номер, равный CountOfClusters+1 (как сказано выше, нумерация ведется с нуля, и в начало таблицы добавлены два резервных элемента).
- Последний сектор FAT должен вычисляться по значению CountOfClusters+1. Использовать с этой целью значения BPB_FATSz16 и BPB_FATSz32 нельзя.

Каталоги файлов

Каталог файлов представляет собой массив 32-байтных элементов — описателей файлов. С точки зрения операционной системы все

каталоги (кроме корневого каталога в системах FAT12 и FAT16) выглядят как файлы и могут содержать произвольное количество записей.

Корневой каталог (Root Directory) — это главный каталог диска, с которого начинается дерево подкаталогов. Для корневого каталога в FAT12 и FAT16 выделено в системной области логического диска специальное место фиксированного размера (16 Кбайт), рассчитанное на хранение 512 элементов. В системе FAT32 корневой каталог является файлом произвольного размера.

Структура элемента каталога файлов приведена в табл. 6.36. Элемент начинается с 11-байтного поля DIR_Name, содержащего так называемое короткое имя (Short name) файла, по которому операционная система обычно осуществляет поиск файла в каталоге. Короткое имя состоит из двух полей: 8-байтного поля, содержащего собственно имя файла, и 3-байтного поля, содержащего расширение. Если введенное пользователем имя файла короче восьми символов, то оно дополняется пробелами (код пробела — 20h); если введенное расширение короче трех байтов, то оно также дополняется пробелами. Разделительная точка между именем и расширением файла не хранится в структуре данных; она подставляется программами операционной системы после имени файла только при выполнении операций, требующих взаимодействия с пользователем (задание имени файла, вывод списка файлов на экран и т. д.). Кроме того, следует учитывать, что в коротком имени все текстовые символы преобразуются операционной системой в верхний регистр.

Таблица 6.36. Структура элемента каталога

Наименование элемента	Смещение	Размер, байт	Описание
DIR_Name	00h	11	Короткое имя файла
DIR_Attr	0Bh	1	Атрибуты файла
DIR_NTRes[1]	0Ch	1	Поле зарезервировано для Windows NT (должно содержать значение 0)
DIR_CrtTimeTenth[1]	0Dh	1	Поле, уточняющее время создания файла (содержит десятки миллисекунд). Значение поля может находиться в пределах от 0 до 199
DIR_CrtTime[1]	0Eh	2	Время создания файла
DIR_CrtDate[1]	10h	2	Дата создания файла

продолжение ⇗

Таблица 6.36 (*продолжение*)

Наименование элемента	Смещение	Размер, байт	Описание
DIR_LstAccDate[1]	12h	2	Дата последнего обращения к файлу для записи или считывания данных
DIR_FstClasHI[1]	14h	2	Старшее слово номера первого кластера файла
DIR_WrtTime	16h	2	Время выполнения последней операции записи в файл
DIR_WrtDate	18h	2	Дата выполнения последней операции записи в файл
DIR_FstClusLO	1Ah	2	Младшее слово номера первого кластера файла
DIR_FileSize	1Ch	4	Размер файла в байтах (32-разрядное число)

Индекс 1 означает, что поле обрабатывается только в файловой системе FAT32. В системах FAT12 и FAT16 поле считается зарезервированным и содержит значение 0.

Первый байт короткого имени (DIR_Name[0]) выполняет функции признака занятости элемента каталога:

- если DIR_Name[0] = E5h, то элемент каталога свободен (то есть его можно использовать при создании нового файла);
- если DIR_Name[0] = 00h, то элемент каталога свободен и является началом чистой области каталога (после него нет ни одного использованного элемента);
- если DIR_Name[0] = 05h, то следует считать, что в этом байте находится ASCII-символ с кодом 0E5h (символ KANJI японской азбуки).

На использование ASCII-символов в коротком имени накладывается ряд ограничений:

- нельзя использовать символы с кодами меньше 20h (за исключением кода 05h в DIR_Name[0]);
- нельзя использовать символы с кодами 22h, 2Ah, 2Bh, 2Ch, 2Eh, 2Fh, 3Ah, 3Bh, 3Ch, 3Dh, 3Eh, 3Fh, 5Bh, 5Ch, 5Dh, 7Ch;
- нельзя использовать символ пробела (код 20h) в DIR_Name[0].

При задании пользователем имени файла допускается отсутствие расширения, но имя должно содержать по крайней мере один символ.

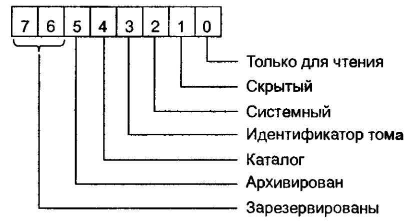


Рис. 6.3. Формат байта атрибутов файла

Формат байта атрибутов файла `DIR_Attr` показан на рис. 6.3. Разряды байта атрибутов устанавливаются в 1 в том случае, если у файла имеется соответствующее свойство:

- бит 0 — файл только для чтения;
- бит 1 — скрытый файл;
- бит 2 — системный файл;
- бит 3 — идентификатор тома;
- бит 4 — каталог;
- бит 5 — архивированный файл;
- биты 6 и 7 — зарезервированы, должны быть установлены в 0.

Признаком того, что свободный элемент каталога используется для хранения участка длинного имени файла, является наличие единиц в разрядах 0–3 байта атрибутов (для описателей файлов и каталогов такое сочетание невозможно).

Час	Минута	Секунда/2
15 11	10 5	4 0

Рис. 6.4. Формат поля времени

Поле времени создания файла `DIR_CrtTime` и поле времени выполнения последней операции записи в файл `DIR_WrtTime` имеют формат, показанный на рис. 6.4. Назначение разрядов полей времени следующее:

- биты 0–4 — двухсекундный отсчет (допустимо значение от 0 до 29);
- биты 5–10 — минута (допустимо значение от 0 до 59);
- биты 11–15 — час (допустимо значение от 0 до 23).

При создании файлов отсчет дат ведется от начала эпохи MS-DOS, то есть от 01.01.1980. Поля даты создания файла DIR_CrtDate, даты последнего обращения к файлу DIR_LstAccDate и даты выполнения последней операции записи в файл DIR_WrtDate имеют формат, показанный на рис. 6.5. Назначение разрядов указанных полей:

- биты 0–4 — день месяца (допустимо значение от 1 до 31);
- биты 5–8 — номер месяца в году (допустимо значение от 1 до 12);
- биты 9–15 — номер года минус 1980 (допустимо значение от 0 до 127).

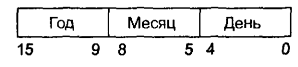


Рис. 6.5. Формат поля даты

Номер первого кластера файла (и точка входа в таблицу FAT) задаются полями DIR_FstClusLO и DIR_FstClusHI, содержащими младшее слово и старшее слово номера кластера соответственно. Для FAT12 и FAT16 определено только поле DIR_FstClusLO, а DIR_FstClusHI является зарезервированным и должно содержать нулевое значение.

Поле размера файла DIR_FileSize является 32-разрядным числом, что позволяет задавать размер файла до 4 Гбайт (4 294 967 295 байт). Хотя каталоги по сути также являются файлами, значение этого поля для них устанавливается в ноль. Ограничение на размер каталога с этим полем никак не связано и вызвано тем, что дисковые утилиты операционной системы используют в качестве счетчика элементов каталога 16-разрядное слово (каталог может содержать до 65 536 32-байтных элементов и занимает, таким образом, не более 2 097 152 байт).

Новые версии операционной системы Windows (начиная с Windows 95) позволяют присваивать файлу (в дополнение к короткому имени) так называемое длинное имя (Long name), используя для его хранения пустые элементы каталога, смежные с основным элементом — описателем файла. Короткое и длинное имена файла являются уникальными, то есть не должны встречаться дважды в одном каталоге.

Информация, которую можно найти о структуре длинного имени в Интернете [74], явно устарела, но некоторые сведения я все-таки получил. Длинное имя записывается не ASCII-символами, а в формате Unicode, где каждому национальному алфавиту соответствует собственный набор кодов. Расплатой за универсальность Unicode

является снижение плотности хранения информации — каждый символ занимает два байта (16-разрядное слово). В пустые элементы каталога длинное имя записывается в разрезанном на кусочки виде, как показано в табл. 6.37.

В одном элементе каталога можно сохранить фрагмент длиной до 13 символов Unicode (поскольку в трех участках имеется в сумме 26 байт). Неиспользованный участок последнего фрагмента заполняется кодами FFFFh.

Таблица 6.37. Структура элемента каталога, хранящего фрагмент длинного имени файла

Наименование элемента	Смещение	Размер, байт	Описание
DIR_Counter	00h	1	Номер фрагмента
DIR_Lname1	01h	10	Первый участок фрагмента имени
DIR_Attr	0Bh	1	Атрибуты файла
DIR_Flags	0Ch	1	Байт флагов
DIR_ChkSum	0Dh	1	Контрольная сумма короткого имени
DIR_LName2	0Eh	12	Второй участок фрагмента имени
DIR_First	1Ah	2	Номер первого кластера (должен быть равен 0)
DIR_LName2	1Ch	4	Третий участок фрагмента имени

Длинное имя записывается в каталог первым, причем фрагменты размещены в обратном порядке, начиная с последнего — как показано на рис. 6.6. Вслед за длинным (полным) именем размещается стандартный описатель файла, содержащий укороченный по специальному алгоритму вариант этого имени. Фрагменты длинного имени пронумерованы. В младших разрядах байта номера фрагмента хранится собственно номер, разряд 6, вероятно, служит признаком последнего фрагмента длинного имени (имеет значение 1 для последнего фрагмента, у остальных фрагментов — 0), а разряд 7 зарезервирован (равен 0).

Все каталоги, за исключением корневого, содержат в двух первых элементах вместо описателей файлов специальные ссылки. В элементе с номером 0 размещается указатель на сам каталог, а в поле имени находится одна точка («.»). В элементе с номером 1 размещается

указатель на родительский каталог (каталог более высокого уровня), а в поле имени находятся две точки («..»). Если ссылка на таблицу FAT у элемента № 1 имеет нулевое значение, то текущий каталог находится в корневом каталоге.

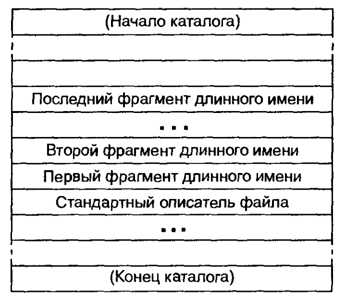


Рис. 6.6. Порядок записи в каталог описателя файла с длинным именем

Организация данных на жестких дисках

Данные на гибком диске организованы в виде одного логического диска, а жесткие диски имеют более сложную структуру. Как уже было указано выше, пространство на жестком диске может быть организовано в виде одного или нескольких разделов, а разделы могут содержать один или несколько логических дисков. Кроме того, разделы могут быть как совместимыми, так и несовместимыми с операционными системами Microsoft, а разделы Microsoft могут иметь различную внутреннюю структуру (ниже будут рассматриваться только разделы со структурой FAT).

Независимо от установленного на диске набора операционных систем (куда могут входить не только системы Microsoft, но и системы других типов), для управления разделами диска используется структура в виде упорядоченного однонаправленного списка. В начальном секторе жесткого диска (в секторе с абсолютным номером 0) размещается главная загрузочная запись (Master Boot Record, сокращенно MBR). В состав MBR входят загрузочная запись (MSB), которая в старой документации именовалась также системно-независимым загрузчиком (NSB), таблица разделов (Partition Table, сокращенно PT) и типовая сигнатура загрузочного сектора (AA55h).

MSB обеспечивает передачу управления загрузочному сектору активного раздела, а MBR содержит список указателей на разделы диска. Внутренняя структура MBR показана в табл. 6.38.

Таблица 6.38. Структура главной загрузочной записи и таблицы разделов

Смещение	Размер поля, байт	Описание
000h	446	Загрузочная запись (MSB)
1BEh	16	Описатель раздела 1
1CEh	16	Описатель раздела 2
1DEh	16	Описатель раздела 3
1EEh	16	Описатель раздела 4
1FEh	2	Сигнатура таблицы разделов (значение AA55h)

Таблица разделов является вершиной дерева разделов (рис. 6.7), разделы в котором обычно размещаются в следующем порядке:

- первичный раздел Microsoft (Primary Partition);
- расширенный раздел Microsoft (Extended Partition);
- разделы других операционных систем (Non-DOS Partitions).

Первичный раздел присутствует на диске в обязательном порядке, а все остальные создаются по усмотрению пользователя. Размеры всех разделов также определяются пользователем в процессе выполнения процедуры начальной разметки диска FDISK. Структура описателя раздела показана в табл. 6.39. Описатель содержит признак активности (то есть того, должна ли операционная система загружаться из данного раздела), координаты начала раздела в формате CHS, код типа раздела (см. табл. 6.26), координаты конца раздела в формате CHS, координаты начала раздела в формате LBA и размер раздела (в 512-байтных секторах).

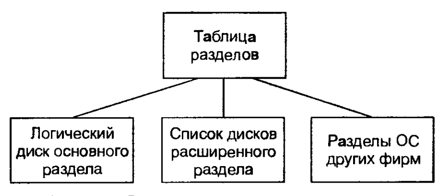


Рис. 6.7. Дерево разделов на жестком диске

Таблица 6.39. Структура описателя раздела

Смещение	Размер поля, байт	Описание
00h	1	Признак активности (0 — раздел не активный, 80h — раздел активный)
01h	1	Номер поверхности диска, с которой начинается раздел
02h	2	Номер цилиндра и номер сектора, с которых начинается раздел[1]
04h	1	Код типа раздела (см. табл. 6.26)
05h	1	Номер поверхности диска, на которой заканчивается раздел
06h	2	Номер цилиндра и номер сектора, которыми заканчивается раздел[1]
08h	4	Абсолютный (логический) номер начального сектора раздела
0Ch	4	Размер раздела (число секторов)

Индекс 1 означает, что номера цилиндра и сектора задаются в формате прерывания Int 13h, то есть биты 0–5 содержат номер сектора, биты 6–7 — старшие два бита 10-разрядного номера цилиндра, биты 8–15 — младшие восемь битов номера цилиндра.

Код раздела используется для определения наличия и положения на диске основного и расширенного разделов (порядок «основной раздел — расширенный раздел — разделы не-DOS» по стандарту Microsoft должен соблюдаться всегда, но на практике возможны разнообразные варианты). После обнаружения нужного раздела его размер и координаты можно извлечь из соответствующих полей описателя. Независимо от используемого физического способа адресации координаты задаются и в формате CHS, и в формате LBA (поскольку функции DOS и BIOS используют оба формата). В табл. 6.40 перечислены только коды разделов Microsoft, а за операционными системами других фирм зарезервированы следующие коды:

- 02h — раздел CP/M;
- 03h — раздел Xenix;
- 07h — раздел OS/2 (файловая система HPFS).

Если в поле кода раздела записан 0, то описатель считается пустым, то есть он не определяет на диске никакого раздела.

Таблица 6.40. Коды разделов операционных систем Microsoft

Код раздела	Вид раздела	Размер	Тип FAT	Ваеден в ОС
01h	Основной	0–15 Мбайт	FAT12	MS-DOS 2.0
04h	Основной	16–32 Мбайт	FAT16	MS-DOS 3.0
05h	Расширенный	0–2 Гбайт	—	MS-DOS 3.3
06h	Основной	32 Мбайт–2 Гбайт	FAT16	MS-DOS 4.0
0Bh	Основной	512 Мбайт–2 Тбайт	FAT32	OSR2
0Ch	Расширенный	512 Мбайт–2 Тбайт	FAT32	OSR2
0Eh	Основной	32 Мбайт–2 Гбайт	FAT16	Windows 95
0Fh	Расширенный	0–2 Гбайт	—	Windows 95

Описатель первичного раздела указывает сразу на загрузочный сектор логического диска (в первичном разделе всегда имеется один и только один диск), а описатель расширенного раздела — на список логических дисков (рис. 6.8), составленный из структур, которые именуются вторичными MBR (Secondary MBR, сокращенно SMBR).

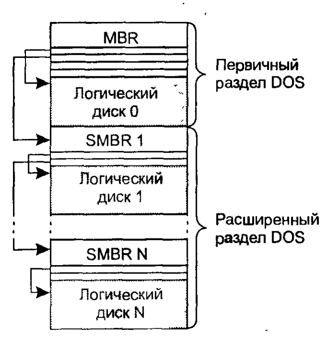


Рис. 6.8. Список разделов DOS на жестком диске

Свой блок SMBR имеется у каждого диска расширенного раздела. SMBR имеет структуру, аналогичную MBR, но загрузочная запись у него отсутствует (заполнена нулями), а из четырех полей описателей используются только два. Первый описатель раздела при этом указывает на логический диск (в поле кода должно стоять значение «основной раздел», соответствующее типу FAT логического диска),

а второй — на следующую структуру SMBR в списке (в поле кода должно стоять какое-либо значение типа «расширенный раздел»). Последний SMBR списка содержит во втором элементе нулевой код раздела.

Чтобы получить доступ к данным, размещенным на логическом диске в расширенном разделе, нужно вначале произвести поиск по списку SMBR. При этом следует учитывать, что операционные системы Microsoft распределяют буквы-имена дисков следующим образом:

- имена A: и B: закреплены за гибкими дисками независимо от того, имеются ли они в данной конфигурации компьютера;
- разделы жестких дисков получают имена, начиная с C:, причем вначале имена получают основные разделы, а уже затем, по второму кругу, именуются логические диски расширенных разделов;
- порядок раздачи имен основным разделам следующий: Master-диск канала 0, Slave-диск канала 0, Master-диск канала 1, Slave-диск канала 1 и т. д.;
- порядок раздачи имен логическим дискам расширенных разделов такой: именуются по порядку все тома Master-диска канала 0, затем тома Slave-диска канала 0, затем Master-диска канала 1, затем Slave-диска канала 1 и т. д.;
- разделы, не имеющие организации типа FAT, просто игнорируются (становятся невидимыми), и имена им не выделяются.

При одновременном использовании нескольких операционных систем следует иметь в виду, что старые системы не распознают разделы, которые имеют новую (введенную после их создания) внутреннюю организацию. Например, MS-DOS версии 6.22 игнорирует логические диски с системой FAT32, то есть не выделяет им имена и не может с ними работать.

Интерфейс ATA

Если возникает потребность в выполнении низкоуровневых операций, то с гибкими дисками удобнее и безопаснее работать через функции BIOS (они более стандартны, чем контроллеры дисководов, разработанные разными фирмами), а с жесткими дисками, наоборот, гораздо проще взаимодействовать напрямую (взаимодействие по стандарту ATA унифицировано лучше, чем функции BIOS у различных изготовителей материнских плат).

Непосредственная работа с регистрами контроллера жесткого диска

Необходимость работать напрямую с регистрами контроллера диска возникает в следующих случаях:

- при переключении процессора в защищенный режим (прерывания DOS и BIOS становятся недоступными);
- при работе с дисками большого (свыше 8 Гбайт) объема или нестандартного (не-DOS) формата;
- в измерительных системах и системах управления реального времени (для ускорения операций ввода-вывода);
- в системах с повышенными требованиями к обеспечению защиты информации (для предотвращения перехвата управления на уровне прерываний, например, компьютерным вирусом).

Обратите еще раз внимание на широкий ассортимент функций в разделе «Группа дисковых функций MS-DOS». Реализуемые ими операции необходимы для нормального функционирования файловой системы — для обеспечения работы в защищенном режиме программист вынужден сам писать нечто аналогичное. Поэтому выполнение перехода в защищенный режим при помощи самодельного программного обеспечения имеет смысл только в случае острой необходимости — например, если требуется реализовать многозадачность, но нельзя применять стандартные операционные системы.

Стандарт АТА [62, 63] позволяет подключать к каждому каналу (то есть к одному кабелю) по два устройства. Современные IBM-подобные персональные компьютеры допускают использование до четырех каналов — следовательно, в общей сложности можно установить на компьютер до 8 дисководов с интерфейсом этого типа. Однако встроенный контроллер материнской платы поддерживает только два канала. Еще два дополнительных канала могут быть реализованы на платах расширения (например, для подключения устройства чтения компакт-дисков через звуковую карту), однако следует иметь в виду, что системный BIOS не будет с ними работать, в том числе — не будет выполнять процедуру поиска подключенных к ним устройств при включении питания. Для работы с дополнительными каналами нужно устанавливать специальные драйверы.

Наличие на материнской плате двух каналов для подключения АТА-устройств фактически стало стандартом. Для этих каналов жестко закреплен диапазон адресов ввода-вывода и номера используемых прерываний (каналы 1 и 2 в табл. 6.41). С дополнительными каналами ситуация менее определенная: фирмы-изготовители догово-

рились между собой о диапазоне адресов регистров, однако номера прерываний не закреплены жестко, они являются рекомендованными — изготовитель, пользователь или операционная система могут вместо них использовать любые другие свободные номера [37].

Таблица 6.41. Пространство ввода-вывода дисководов ATA

Номер канала	Диапазон адресов		Номер сигнала прерывания IRQ
	Регистры команды	Регистры контроля	
1	1F0h–1F7h	3F6h–3F7h	14
2	170h–177h	376h–377h	15
3	1E8h–1EFh	3Eeh–3EFh	11
4	168h–16Fh	36Eh–36Fh	10

В табл. 6.42 даны адреса регистров для двух основных каналов (для дополнительных каналов порядок регистров аналогичный). Для адресации данных на диске с интерфейсом ATA можно использовать либо режим LBA, либо режим CHS, причем назначение регистров контроллера зависит от используемого режима адресации. Все регистры, за исключением регистра данных, 8-разрядные, а регистр данных — 16-разрядный.

Таблица 6.42. Функциональное назначение регистров контроллера жесткого диска

Адрес регистра		Назначение регистра	
Канал 1	Канал 2	В режиме чтения	В режиме записи
1F0h	170h	Регистр данных (DR)	
1F1h	171h	Регистр ошибок (ER)	Регистр свойств (FR)
1F2h	172h	Счетчик секторов (SC)	
1F3h	173h	В режиме CHS — регистр номера сектора (SN); в режиме LBA — разряды 0–7 адреса сектора	
1F4h	174h	В режиме CHS — регистр младшего байтв номера цилиндра (CL); в режиме LBA — разряды 8–15 адреса сектора	
1F5h	175h	В режиме CHS — регистр старшего байта номера цилиндра (CH);	

Адрес регистра		Назначение регистра	
Канал 1	**Канал 2**	**В режиме чтения**	**В режиме записи**
		в режиме LBA — разряды 16–23 адреса сектора	
1F6h	176h	В режиме CHS — регистр номера устройства и головки (DH); в режиме LBA — номер устройства и разряды 24–27 адреса сектора	
1F7h	177h	Регистр состояния (SR)	Регистр команд (CR)
3F6h	376h	Альтернативный регистр состояния (AC)	Регистр управления устройством (DC)
3F7h	377h	Не используется	

Рассмотрим формат регистров более подробно. Регистр данных (DR) используется при выполнении операции чтения или записи сектора в программном режиме ввода-вывода) (PIO). Этот регистр недоступен, пока не начнется операция чтения или записи. Нельзя обращаться к регистру, когда происходит обмен информацией между диском и памятью в режиме прямого доступа (DMA).

Передача данных через регистр осуществляется 16-разрядными словами. Обратите внимание: это 16-разрядный регистр и его адресное пространство перекрывает следующий за ним регистр ошибок (еще один радиолюбительский трюк под названием «экономия пространства ввода-вывода»). Даже в том случае, если старший байт данных не используется (такая ситуация возможна в некоторых командах), чтение/запись все равно нужно выполнять целыми словами (обнуляя старший байт перед операцией записи и после операции чтения).

Специально для упрощения и ускорения работы с регистром данных контроллера жесткого диска в набор команд процессоров с архитектурой Intel x86 (начиная с 80186) были введены операции группового ввода-вывода INSW и OUTSW, хотя можно работать с данными и при помощи обычных команд ввода-вывода IN и OUT.

Рассмотрим более подробно регистры контроллера жесткого диска. Регистр ошибок (ER) доступен только для чтения. Он определяет состояние адаптера после выполнения операции. Содержимое этого регистра нужно проверять в двух случаях:

- после выполнения любой команды, если установлен бит ошибки ERR в регистре состояния;
- после выполнения команды «диагностика» или после выполнения внутренней диагностики адаптера по системному сбросу.

Коды регистра ошибок после выполнения команды «диагностика»:

- 01h — нет ошибки: устройство 0 исправно, устройство 1 — либо исправно, либо не присутствует в системе (не подключено);
- 00, 02h–7Fh — устройство 0 неисправно, устройство 1 — либо исправно, либо не подключено;
- 81h — устройство 0 исправно, устройство 1 неисправно;
- 80h, 82h–FFh — оба устройства (0 и 1) неисправны.

Во всех остальных случаях разряды регистра ошибок, формат которого показан на рис. 6.9, имеют следующее назначение:

- бит 0 (AMNF) — не найден адресный маркер сектора;
- бит 1 (TKONF) — не найдена нулевая дорожка при выполнении команды «рекалибровка»;
- бит 2 (ABRT) — аварийное прекращение выполнения команды;
- бит 3 (MCR) — получен запрос на смену носителя информации;
- бит 4 (IDNF) — сектор с заданными координатами (цилиндр, головка, сектор) не найден;
- бит 5 (MC) — произведена смена носителя информации;
- бит 6 (UNC) — некорректируемая ошибка данных;
- бит 7 — зарезервирован.

X	UNC	MC	IDNF	MCR	ABRT	TKONF	AMNF
7	6	5	4	3	2	1	0

Рис. 6.9. Формат регистра ошибок

Если при выполнении команды ошибок не было, то все разряды регистра ошибок содержат нули. Если при выполнении команды происходит какая-либо ошибка, то соответствующий разряд регистра устанавливается в 1.

Регистр свойств (FR) расположен по тому же адресу, что и регистр ошибок, но доступен только для записи. Формат данных регистра изменяется в зависимости от команды. При выполнении обычных операций ввода-вывода этот регистр не применяется.

В регистр счетчика секторов (SC) заносится количество секторов, которое должно быть считано или записано (при записи 0 в этот регистр происходит обработка 256 секторов). Значение этого регистра уменьшается на единицу после обработки каждого сектора. При выполнении мультисекторной операции сектора должны располагаться на диске последовательно, друг за другом (то есть область дан-

ных должна быть непрерывной). Этот регистр доступен не только для записи, но и для считывания — в случае возникновения ошибки при выполнении операции чтения или записи в этом регистре будет находиться число секторов, оставшихся необработанными.

В регистр номера сектора (SN) в режиме CHS загружается стартовый номер сектора при операциях чтения/записи. После обработки каждого сектора в этот регистр автоматически заносится номер следующего сектора, подлежащего обработке. Регистр доступен для чтения/записи. После выполнения команды он содержит номер последнего обработанного сектора. В режиме LBA регистр номера сектора содержит младший байт линейного адреса сектора (разряды 0–7).

Регистры младшего (CL) и старшего (CH) байтов номера цилиндра в режиме CHS определяют стартовый цилиндр для выполнения команды (в старых контроллерах дисков использовалось только 2 младших разряда регистра CH, то есть максимальный номер цилиндра был равен 1023). Регистры доступны для чтения/записи. После выполнения команды они содержат текущий адрес цилиндра.

В режиме LBA регистр CL содержит разряды 8–15, а регистр CH — разряды 16–23 линейного адреса сектора.

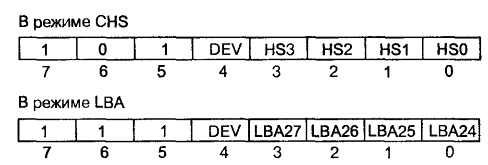


Рис. 6.10. Формат регистра номера устройства и головки

Формат регистра номера устройства и головки (DH) показан на рис. 6.10. Регистр доступен для чтения и записи. В режиме CHS разряды регистра имеют следующие значения:

- биты 0–3 (HS0–HS3) — номер головки;
- бит 4 (DEV) — выбор устройства (0 или 1);
- бит 5 — зарезервирован (должен быть установлен в 1);
- бит 6 (LBA) — признак режима LBA (должен быть сброшен в 0);
- бит 7 — зарезервирован (должен быть установлен в 1).

В режиме LBA назначение разрядов следующее:

- биты 0–3 (LBA24–LBA27) — разряды 24–27 линейного адреса сектора;

- бит 4 (DEV) — выбор устройства (0 или 1);
- бит 5 — зарезервирован (должен быть установлен в 1);
- бит 6 (LBA) — признак режима LBA (должен быть установлен в 1);
- бит 7 — зарезервирован (должен быть установлен в 1).

BSY	DRDY	DF	DSC	DRQ	CORR	IDX	ERR
7	6	5	4	3	2	1	0

Рис. 6.11. Формат регистра состояния

Формат регистра состояния (SR) показан на рис. 6.11. Регистр состояния отображает состояние устройства и доступен только для чтения. Значения битов регистра состояния (возникновение определенного состояния индицируется установкой соответствующего бита в 1) перечислены ниже:

- бит 0 (ERR) — при выполнении команды произошла ошибка (этот бит сбрасывается при поступлении следующей команды или аппаратном сбросе устройства). В случае возникновения ошибки информацию о ней можно получить из регистра ошибок;
- бит 1 (IDX) — сигнал Index, каждым производителем трактуется по-своему (поэтому при работе с диском этот сигнал нужно просто игнорировать);
- бит 2 (CORR) — при считывании с диска имела место ошибка, но данные были успешно скорректированы;
- бит 3 (DRQ) — устройство готово к обмену данными с процессором;
- бит 4 (DSC) — головки чтения/записи завершили поиск заданного сектора;
- бит 5 (DF) — устройство неисправно;
- бит 6 (DRDY) — устройство готово к приему следующей команды;
- бит 7 (BSY) — устройство занято выполнением какой-то операции, ему нельзя передавать команды или данные, нельзя считывать содержимое регистров (во избежание получения ложных данных).

ПРИМЕЧАНИЕ

Если разряд готовности к приему команды DRDY сброшен в 0, устройство реагирует только на две аварийные команды — выполнить диагностику устройства (EXECUTE DEVICE DIAGNOSTIC) и установить параметры устройства (INITIALISE DEVICE PARAMETERS).

Регистр команд (CR) используется для загрузки кода выполняемой команды. Запись кода команды должна производиться в последнюю очередь — только после того, как в остальные регистры занесены все необходимые для ее выполнения данные. Выполнение команды начинается сразу после записи кода в этот регистр.

Альтернативный регистр состояния (AC) по формату аналогичен регистру состояния (SR), но считывание данных из него не приводит к снятию запроса прерывания.

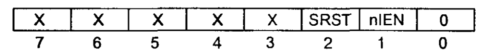


Рис. 6.12. Формат регистра управления устройством

Регистр управления устройством (DC) доступен только для записи. Значения битов этого регистра следующие (см. рис. 6.12):

- бит 0 — не используется (всегда должен быть сброшен в 0);
- бит 1 (nIEN) — запрет прерывания (0 — прерывание разрешено, 1 — запрещено);
- бит 2 (SRST) — программный сброс всех подключенных к данному каналу устройств (сброс происходит при установке этого бита в 1);
- биты 3–7 — зарезервированы (игнорируются).

Коды обязательных команд ATA

В соответствии со стандартом команды интерфейса ATA делятся на три основные группы:

- обязательные (Mandatory) команды;
- дополнительные (Optional) команды;
- специфические команды изготовителя (Vendor specific implementation).

Как и в случае с устройствами других типов, программисты, не связанные непосредственно с изготовителями оборудования, не имеют доступа к полной документации и вынуждены ограничиваться командами первой группы, полностью определенными в стандарте. В таблице 6.43 дано описание команд, необходимых для работы с современными дисководами и потому обязательных для всех выпускаемых устройств (многие команды уже либо устарели, либо могут применяться только изготовителями дисков, и программистам

их использовать не рекомендуется). Для регистров младшего и старшего байта номера цилиндра в графе «Используемые регистры» применяются общее обозначение CY. В графе регистра номера устройства и головки DH обозначение D говорит о том, что используется только номер устройства, а информация о номере головки игнорируется; обозначение D* показывает, что, хотя команда адресована устройству 0, выполняют ее оба устройства.

В графе «Протокол» используются следующие обозначения:

- DM — обмен данными в режиме DMA;
- ND — команда не требует обмена данными;
- PI — ввод данных в режиме PIO;
- PO — вывод данных в режиме PIO.

Таблица 6.43. Коды команд основных дисковых операций

Команда	Код	Протокол	Используемые регистры				
			FR	SC	SN	CY	DH
EXECUTE DEVICE DIAGNOSTIC — выполнить диагностику устройства	90h	ND					D*
IDENTIFY DEVICE — идентифицировать устройство	ECh	PI					D
INITIALIZE DEVICE PARAMETERS — установить параметры устройства	91h	ND		Да			Да
READ DMA (w/ retry) — чтение в режиме DMA с повторами	C8h	DM		Да	Да	Да	Да
READ DMA (w/o retry) — чтение в режиме DMA без повторов	C9h	DM		Да	Да	Да	Да
READ MULTIPLE — чтение группы секторов	C4h	PI		Да	Да	Да	Да
READ SECTOR(S) (w/ retry) — чтение секторов с повторами	20h	PI		Да	Да	Да	Да
READ SECTOR(S) (w/o retry) — чтение секторов без повторов	21h	PI		Да	Да	Да	Да
READ VERIFY SECTOR(S) (w/ retry) — контрольное чтение секторов с повторами	40h	PI		Да	Да	Да	Да

Команда	Код	Протокол	Используемые регистры				
			FR	SC	SN	CY	DH
READ VERIFY SECTOR(S) (w/o retry) — контрольное чтение секторов без повторов	41h	PI		Да	Да	Да	Да
SEEK — поиск сектора	70h	ND			Да	Да	Да
SET FEATURES — установка свойств устройства	EFh	ND	Да				D
SET MULTIPLE MODE — установить параметры множественного режима	C6h	ND		Да			D
WRITE DMA (w/ retry) — запись в режиме DMA с повторами	CAh	DM		Да	Да	Да	Да
в режиме DMA без повторов	CBh	DM		Да	Да	Да	Да
WRITE MULTIPLE — запись группы секторов	C5h	PO		Да	Да	Да	Да
WRITE SECTOR(S) (w/ retry) — запись секторов с повторами	30h	PO		Да	Да	Да	Да
WRITE SECTOR(S) (w/o retry) — запись секторов без повторов	31h	PO		Да	Да	Да	Да

Далеко не все команды из табл. 6.29 нужны рядовому пользователю. Следует особо отметить подгруппу команд, которая реально применяется при работе с дисками: IDENTIFY DEVICE, READ DMA (w/ retry), READ SECTOR(S) (w/ retry), WRITE DMA (w/ retry), WRITE SECTOR(S) (w/ retry). Первая команда из данного списка позволяет проверить наличие устройства и определить его параметры, а остальные служат для ввода и вывода данных.

Примеры использования базовых команд ATA будут рассмотрены в конце главы. Отметим, что команду IDENTIFY DEVICE нужно использовать только при запуске программы, непосредственно работающей с жестким диском, чтобы удостовериться в том, что он подключен к контроллеру и может работать в тех режимах, на которые рассчитана программа — например, многие старые диски объемом до 500 Мбайт не поддерживают режим LBA. При выполнении команды IDENTIFY DEVICE диск выдает всю информацию о себе в виде структуры данных из 256 слов по 16 разрядов, которая описана в табл. 6.44.

Таблица 6.44. Информация об устройстве, выдаваемая по команде IDENTIFY DEVICE

Номер х слоаа	F/V	Описание
0	F	Общая информация о конфигурации устройства:
		бит 0 — зарезервирован;
		биты 1–5 — определяются изготовителем;
		бит 6 — устройство с несменным носителем, если данный разряд установлен в 1;
		бит 7 — устройство со сменным носителем, если данный разряд установлен в 1;
		биты 8–14 — определяются изготовителем;
		бит 15 — тип интерфейса (0 — устройство АТА)
1	F	Число логических цилиндров
2	R	Зарезервировано
3	F	Число логических головок
4	X	Определяется изготовителем
5	X	Определяется изготовителем
6	F	Число логических секторов на дорожке
7–9	X	Определяются изготовителем
10–19	F	Серийный номер (20 ASCII-символоа)
20	X	Определяется изготовителем
21	X	Определяется изготовителем
22	F	Не используется (устарело)
23–26	F	Версия встроенного в ПЗУ устройства программного обеспечения (8 ASCII-символов)
27–46	F	Номер модели устройства (40 ASCII-символов)
47	F	Биты 0–7 — значение 0 зарезервировано, значения 01h–FFh соответствуют максимальному количеству секторов, которое можно передавать в командах группового чтения и записи;
	X	биты 8–15 — определяются изготовителем
48	R	Зарезервировано
49		Возможности устройства:
	X	биты 0–7 — определяются изготовителем;
	R	биты 8–9 — зарезервированы, но должны быть установлены в 1;
	F	бит 10 — если в данном разряде установлена 1, то сигнал IORDY может быть запрещен;
	F	бит 11 — поддержка IORDY (0 — может отсутствовать, 1 — присутствует);

Номер слова	F/V	Описание
	R	бит 12 — зарезервирован;
	F	бит 13 — способ задания значений таймера Standby (0 — значения задаются изготовителем устройства, 1 — значения задаются по правилам интерфейса ATA);
	R	биты 14–15 — зарезервированы
50	F	Возможности устройства.
		бит 0 — имеет значение 1;
		биты 1–13 — зарезервированы;
		бит 14 — имеет значение 1;
		бит 15 — имеет значение 0
51	X	Биты 0–7 — определяются изготовителем;
	F	биты 8–15 — номер используемого режима передачи PIO
52	R	Зарезервировано
53	V	Бит 0 — значения в словах 54–58: 0 — недействительны; 1 — действительны;
	F	бит 1 — значения в словах 64–70: 0 — недействительны; 1 — действительны;
	F	бит 2 — значение слова 88:
		0 — недействительно;
		1 — действительно;
	R	биты 3–15 зарезервированы
54	V	Текущее число логических цилиндров
55	V	Текущее число логических головок
56	V	Текущее число логических секторов на дорожке
57–58	V	Текущая емкость в секторах
59	V	Биты 0–7 — текущее количество секторов, которое может быть передано за одно прерывание в групповых операциях чтения/записи;
	V	бит 8 — параметры групповых операций действительны, если данный разряд установлен в 1;
	R	биты 9–15 — зарезервированы
60–61	F	Общее число секторов, адресуемых пользователем в режиме LBA
62	R	Зарезервировано
63		Поддерживаемые режимы Multiword DMA (режим поддерживается, если соответствующий разряд установлен в 1):
	F	бит 0 — поддержка режима Multiword DMA 0;

продолжение ⇗

Таблица 6.44 (*продолжение*)

Номер х слова	F/V	Описание
	F	бит 1 — поддержка режима Multiword DMA 1;
	F	бит 2 — поддержка режима Multiword DMA 2;
	R	биты 3–7 зарезервированы
		Выбор используемого режима Multiword DMA (режим активен, если соответствующий разряд установлен в 1):
	V	бит 8 — выбран режим Multiword DMA 0;
	V	бит 9 — выбран режим Multiword DMA 1;
	V	бит 10 — выбран режим Multiword DMA 2;
	R	биты 3–7 зарезервированы
64	F	Поддерживаемые улучшенные режимы PIO (режим поддерживается, если соответствующий бит установлен в 1):
		бит 0 — поддержка режима 3;
		бит 1 — поддержка режима 4;
		биты 2–7 зарезервированы для последующих версий улучшенных режимов PIO;
	R	биты 8–15 — зарезервированы
65	F	Минимальное время цикла передачи слова Multiword DMA в наносекундах
66	F	Рекомендованное изготовителем время цикла передачи Multiword DMA в наносекундах
67	F	Минимальное время цикла передачи PIO (без проверки сигнала готовности) в наносекундах
68	F	Минимальное время цикла передачи PIO (с проверкой сигнала готовности) в наносекундах
69–70	R	Зарезервированы для будущих команд, реализующих «очереди» и «перекрытие»
71–74	R	Зарезервированы
75	F	Глубина очереди:
		биты 0–4 — максимально допустимвя глубина очереди;
		биты 5–15 зарезервированы
76–79	R	Зарезервированы
80	F	Основной номер версии ATA (номер версии соответствует номеру бита, значение которого установлено в 1; если данное слово содержит код 0000h или код FFFFh, то его значение не действительно):
		бит 0 зарезервирован;
		бит 1 — поддерживается ATA-1;

Номер слова	F/V	Описание
		бит 2 — поддерживается ATA-2;
		бит 3 — поддерживается ATA-3;
		бит 4 — поддерживается ATA/ATAPI-4; бит 5 — поддерживается ATA/ATAPI-5;
		бит 6 — поддерживается ATA/ATAPI-6;
		бит 7 зарезервирован для ATA/ATAPI-7; бит 8 зарезервирован для ATA/ATAPI-8;
		бит 9 зарезервирован для ATA/ATAPI-9;
		бит 10 зарезервирован для ATA/ATAPI-10;
		бит 11 зарезервирован для ATA/ATAPI-11;
		бит 12 зарезервирован для ATA/ATAPI-12;
		бит 13 зарезервирован для ATA/ATAPI-13;
		бит 14 зарезервирован для ATA/ATAPI-14;
		бит 15 зарезервирован
81	F	Номер реализации версии ATA (если в поле записано значение 0000h или FFFFh, то данное поле не действительно)
82	F	Поддерживаемые устройством «свойства» и группы команд («свойство» или группа команд поддерживается, если соответствующий разряд установлен в 1; если в словах 82 и 83 записаны значения 0000h или FFFFh, то данное поле не действительно):
		бит 0 — поддерживается набор команд Smart;
		бит 1 — поддерживается набор команд секретности;
		бит 2 — поддерживается набор команд для устройств со сменными носителями;
		бит 3 — поддерживается набор команд управления энергопотреблением;
		бит 4 — поддерживается набор пакетных команд;
		бит 5 — разрешено кэширование при записи;
		бит 6 — разрешено упреждение;
		бит 7 — разрешено прерывание при освобождении шины;
		бит 8 — разрешено прерывание SERVICE;
		бит 9 — поддерживается команда DEVICE RESET;
		бит 10 — поддерживается защищенная область хоста;
		бит 11 — не используется (устарел);
		бит 12 — поддерживается команда WRITE BUFFER;
		бит 13 — поддерживается команда READ BUFFER;

продолжение ⇨

Таблица 6.44 (*продолжение*)

Номер слова	F/V	Описание
		бит 14 — поддерживается команда NOP;
		бит 15 — не используется (устарел)
83	F	Поддерживаемые устройством «свойства» и группы команд («свойство» или группа команд поддерживается, если соответствующий разряд установлен в 1; если в словах 82 и 83 записаны значения 0000h или FFFFh, то данное поле недействительно):
		бит 0 — поддерживается команда DOWNLOAD MICROCODE;
		бит 1 — поддерживаются очереди в режиме DMA;
		бит 2 — поддерживается флэш-память;
		бит 3 — поддерживается «улучшенное» управление энергопотреблением;
		бит 4 — поддерживается извещение о состоянии сменного носителя;
		биты 5–13 зарезервированы;
		бит 14 должен быть установлен в 1;
		бит 15 должен быть сброшен в 0
84	F	Признак наличия поддержки расширенного набора команд и свойств (если в словах 82, 83 и 84 записаны значения 0000h или FFFFh, то данное поле недействительно):
		биты 0–13 зарезервированы;
		бит 14 должен быть установлен в 1;
		бит 15 должен быть сброшен в 0
85	V	Разрешенные «свойства» и группы команд («свойство» или группу команд разрешено использовать, если соответствующий разряд установлен в 1; если в словах 85, 86 и 87 записаны значения 0000h или FFFFh, то данное поле недействительно):
		бит 0 — разрешено использовать набор команд Smart;
		бит 1 — разрешено использовать набор команд секретности;
		бит 2 — разрешено использовать набор команд для устройств со сменными носителями;
		бит 3 — разрешено использовать набор команд управления энергопотреблением;
		бит 4 — разрешено использовать набор пакетных команд;

Номер слова	F/V	Описание
		бит 5 — разрешено использовать кэширование при записи;
		бит 6 — разрешено использовать упреждение;
		бит 7 — разрешено использовать прерывание при освобождении шины;
		бит 8 — разрешено использовать прерывание SERVICE;
		бит 9 — разрешено использовать команду DEVICE RESET;
		бит 10 — разрешено использовать защищенную область хоста;
		бит 11 — не используется (устарел);
		бит 12 — разрешено использовать команду WRITE BUFFER;
		бит 13 — разрешено использовать команду READ BUFFER;
		бит 14 — разрешено использовать команду NOP;
		бит 15 — не используется (устарел)
86	V	Разрешенные «свойства» и группы команд («свойство» или группу команд разрешено использовать, если соответствующий разряд установлен в 1; если в словах 85, 86 и 87 записаны значения 0000h или FFFFh, то данное поле недействительно):
		бит 0 — разрешено использовать команду DOWNLOAD MICROCODE;
		бит 1 — разрешено использовать очереди в режиме DMA;
		бит 2 — разрешено использовать команды, предназначенные для работы с флэш-памятью;
		бит 3 — разрешено использовать «улучшенное» управление энергопотреблением;
		бит 4 — разрешено использовать извещение о состоянии сменного носителя;
		биты 5–15 зарезервированы
87	V	Признак достоверности слов 85 и 86 (если в словах 85, 86 и 87 записаны значения 0000h или FFFFh, то данное поле недействительно):
		биты 0–13 зарезервированы;
		бит 14 должен быть установлен в 1;
		бит 15 должен быть сброшен в 0
88		Признаки наличия поддержки и использования режимов Ultra DMA (установка разряда в 1 означает наличие соответствующего свойства):

продолжение ⇒

Таблица 6.44 (*продолжение*)

Номер x слова	F/V	Описание
	F	бит 0 — поддержка режима Ultra DMA 0;
	F	бит 1 — поддержка режима Ultra DMA 1;
	F	бит 2 — поддержка режима Ultra DMA 2;
	R	биты 3–7 зарезервированы;
	V	бит 8 — выбран режим Ultra DMA 0;
	V	бит 9 — выбран режим Ultra DMA 1;
	V	бит 10 — выбран режим Ultra DMA 2;
	R	биты 11–15 зарезервированы
89	F	Время, необходимое для очистки (стирания) логического устройства в режиме секретности (время задается в минутах и вычисляется путем умножения значения данного поля на 2; код 00h означает, что значение времени очистки не определено)
90	F	Время, необходимое для «улучшенной» очистки логического устройства в режиме секретности (время задается в минутах и вычисляется путем умножения значения данного поля на 2; код 00h означает, что значение времени «улучшенной» очистки не определено)
91	V	Текущее значение кода управления энергопотреблением
92~126	R	Зарезервировано
127	F	Поддержка извещения о состоянии сменного носителя: биты 0–1 — код способа поддержки извещения о состоянии сменного носителя (00b — извещение о состоянии носителя не поддерживается устройством, 00b — извещение поддерживается, коды 10b и 11b зарезервированы); биты 2–15 зарезервированы
128	F	Статус секретности: бит 0 — поддержка секретности (0 — отсутствует, 1 — имеется); бит 1 — использование секретности (0 — запрещено, 1 — разрешено); бит 2 — блокировка режима секретности (0 — отсутствует, 1 — имеется); бит 3 — приостановка режима секретности (0 — отсутствует, 1 — имеется); бит 4 — счетчик секретности (0 — отсутствует, 1 — имеется);

Номер слова	F/V	Описание
		бит 5 — поддержка улучшенного режима стирания (0 — отсутствует, 1 — имеется);
		биты 6-7 зарезервированы;
		бит 8 — уровень секретности (0 — высокий, 1 — максимальный);
		биты 9-15 зарезервированы
129-159	X	Определяется изготовителем
160-255	R	Зарезервировано

В колонке «F/V» табл. 6.44, которая показывает, является ли значение поля константой или может изменяться, используются следующие сокращенные обозначения:

- F — значение поля фиксировано;
- V — значение поля может изменяться в зависимости от состояния устройства и выполняемой команды;
- R — поле зарезервировано и должно содержать нулевое значение;
- X — значение поля является неопределенным (зависит от изготовителя).

Номера слов в таблице даны в десятичном коде. Чтобы вычислить смещение какого-либо нужного вам поля данных от начала таблицы, нужно умножить номер начального слова этого поля на 2.

Режимы и протоколы передачи информации

Обмен информацией между жестким диском, обозначенным в стандарте термином Device (устройство), и компьютером, обозначенным как Host (хост), должен выполняться по строго определенным правилам. Из таблицы 6.29 видно, что для каждой команды в стандарте определены режим и протокол обмена информацией между устройством и хостом.

Для обмена данными между диском и контроллером используются 16-разрядные слова в пакетах по 256 штук. При работе с современными жесткими дисками может применяться один из двух основных режимов передачи данных: режим программного ввода-вывода (PIO) через процессор или режим прямого доступа к памяти (DMA). Оба режима постоянно совершенствуются и в настоящее

время имеют несколько вариантов реализации, отличающихся все более высокой скоростью передачи данных между контроллером и устройством, как показано в табл. 6.45 (чем современнее реализация режима, тем выше его порядковый номер).

Таблица 6.45. Скорость передачи данных в различных режимах ввода-вывода (Мбайт/с)

Номер режима	PIO	Single word DMA	Multi-word DMA	Ultra DMA
0	3,3	2,08	4,12	16,7
1	5,2	4,16	13,3	25
2	8,3	8,33	16,7	33,3
3	11,1	—	—	44,4
4	16,7	—	—	66,7
5	—	—	—	100

Современные дисковые накопители поддерживают по несколько различных скоростей передачи данных для режимов PIO и DMA, чтобы обеспечить совместимость как со старыми, так и с новыми материнскими платами и контроллерами. В процессе тестирования оборудования при загрузке компьютера BIOS опрашивает подключенные к контроллеру устройства и автоматически настраивает их на самые скоростные режимы, какие только могут быть реализованы контроллером и дисками.

Некоторые старые контроллеры имели свойство настраиваться на скорость самого медленного из подключенных к каналу устройств, поэтому при их использовании не рекомендовалось подсоединять к одному кабелю жесткий диск и устройство чтения компакт-диска. Сейчас ситуация изменилась: современные контроллеры допускают подключение к одному каналу и быстрых, и медленных устройств, а дисководы компакт-дисков обеспечивают почти такую же скорость передачи данных, как и жесткие диски.

Для каждой АТА-команды стандарт определяет протокол, но не задает скорость передачи данных (см. табл. 6.29). Стандарт также приводит (в виде диаграмм) описание типовых протоколов. Протокол ввода данных в режиме PIO показан на рис. 6.13 и 6.14, протокол вывода данных в режиме PIO — на рис. 6.15 и 6.16, протокол для команд, не требующих передачи данных — на рис. 6.17, протокол передачи данных в режиме DMA — на рис. 6.18 и 6.19.

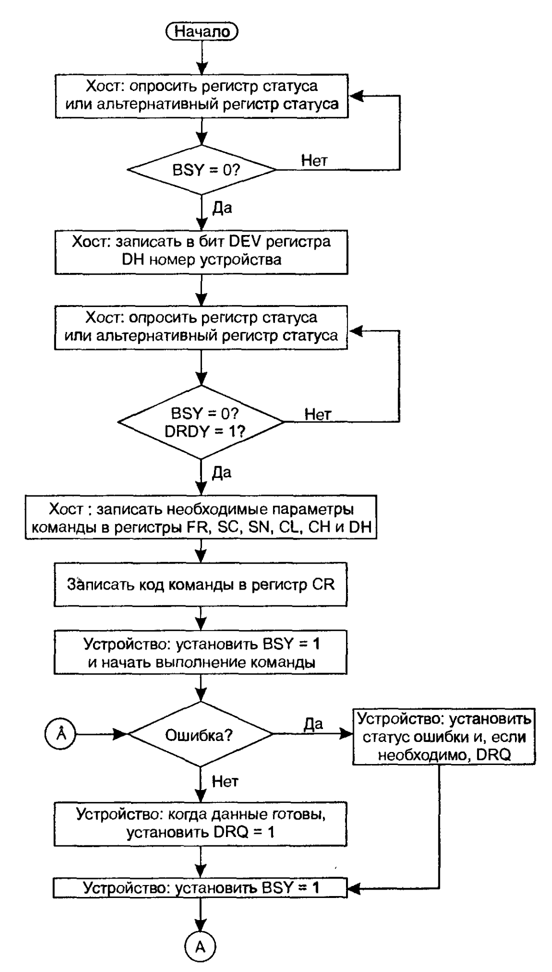


Рис. 6.13. Типовой протокол ввода данных с диска в режиме PIO (начало)

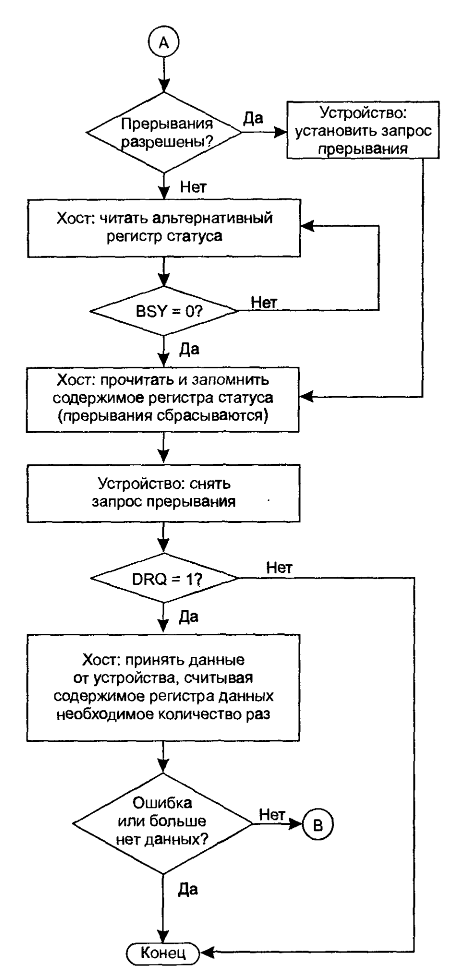


Рис. 6.14. Типовой протокол ввода данных с диска в режиме PIO (окончание)

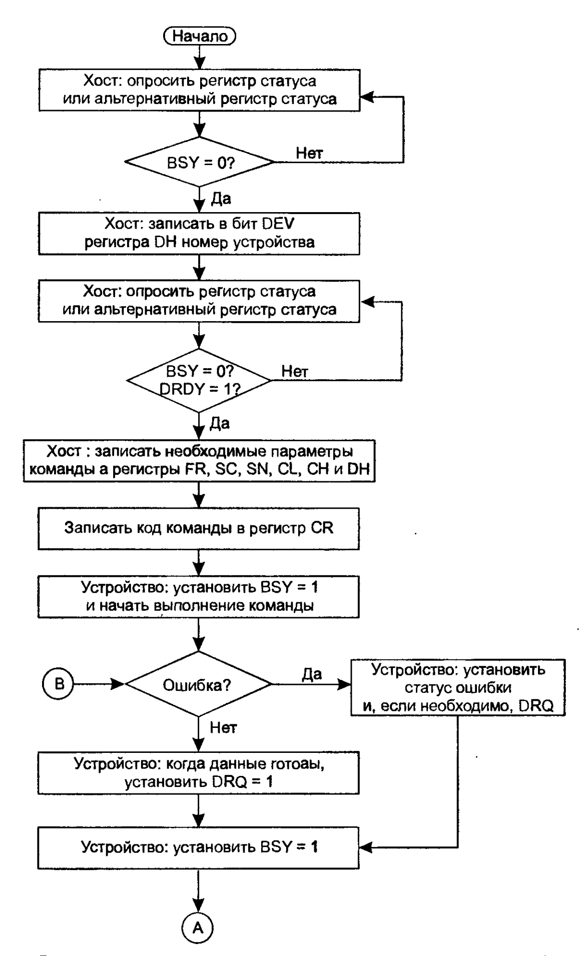


Рис. 6.15. Типовой протокол вывода данных с диска в режиме PIO (начало)

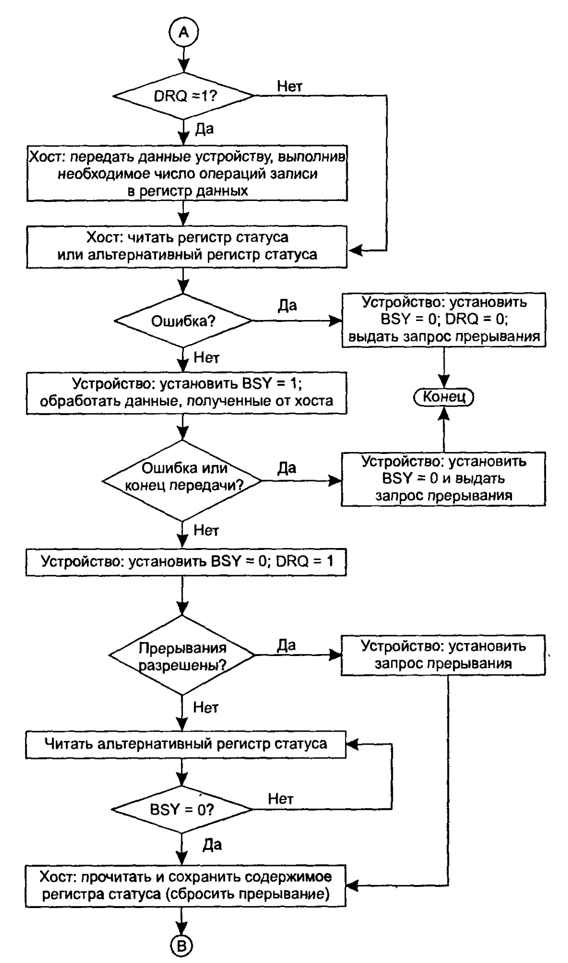


Рис. 6.16. Типовой протокол вывода данных с диска в режиме PIO (окончание)

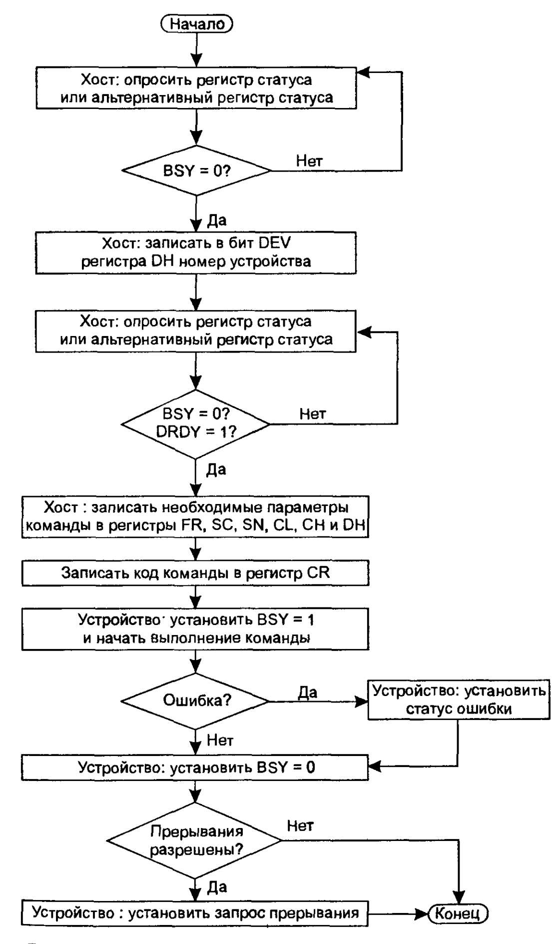


Рис. 6.17. Типовой протокол выполнения операций, не требующих передачи блока данных

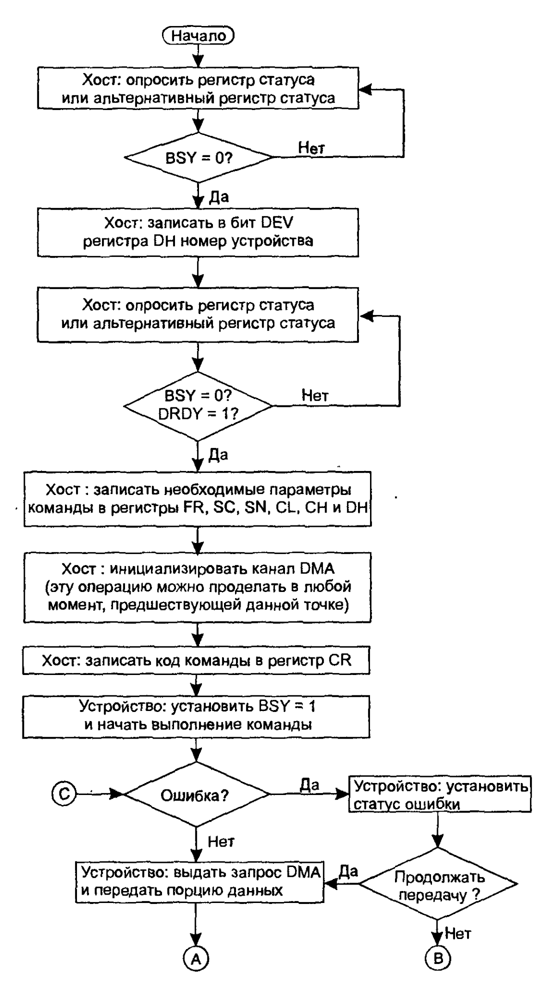


Рис. 6.18. Типовой протокол передачи данных в режиме DMA (начало)

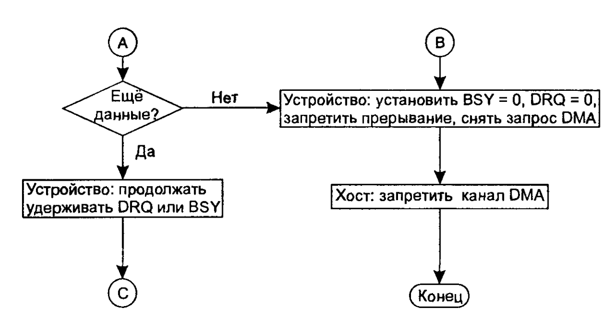


Рис. 6.19. Типовой протокол передачи данных в режиме DMA (окончание)

Приведенные на рис. 6.13–6.19 протоколы не следует воспринимать слишком буквально (как схемы алгоритмов). В стандарте они — видимо, для большей наглядности — были упрощены (например, нет контроля за временем выполнения операций, и если устройство зависнет, то программа может застрять в бесконечном цикле ожидания сигнала готовности). Кроме того, приведенные схемы носят название примерных и подвергаются изменениям в каждой новой версии стандарта ATA/ATAPI.

В программах, предназначенных для работы в MS–DOS, можно ограничиться простыми протоколами ввода-вывода PIO, поскольку значительно более сложный для реализации протокол DMA дает выигрыш по скорости только в многозадачном режиме. Дело в том, что скорость передачи данных между диском и контроллером не является основным лимитирующим фактором — гораздо больше времени тратится на поиск нужной области на диске и считывание или запись информации. DMA приносит пользу только в том случае, если процессору есть, чем заняться, пока диск ищет и передает информацию. Однако для того, чтобы задействовать DMA, нужно правильно запрограммировать контроллер, обеспечивающий диску прямой доступ к памяти, что может оказаться нетривиальной задачей вследствие отсутствия описаний на новые типы DMA-контроллеров. Особую осторожность следует проявлять при программировании режима UltraDMA 66, поскольку он все еще находится в стадии освоения изготовителями периферийного оборудования.

Примеры программ, непосредственно работающих с контроллером жесткого диска

Листинг 6.7 содержит набор процедур, предназначенных для непосредственной работы с регистрами контроллера жесткого диска:

- процедура ReadHDDSector считывает с заданный сектор с указанного диска, используя LBA и протокол ввода PIO;
- процедура SendCommandToHDD предназначена для посылки команды контроллеру диска (она является внутренней вспомогательной подпрограммой данного модуля и не должна вызываться из других модулей программы);
- процедура ReadHDD_ID считывает идентификационный блок данных заданного диска, также используя протокол PIO.

Указанные процедуры используют самые простые способы обмена данными с жестким диском — линейную адресацию LBA и программный ввод-вывод PIO; прерывания, DMA и мультисекторная передача данных не применяются. Для определения готовности устройства к началу обмена информацией применяется метод циклического опроса регистра состояния: вначале проверяется бит занятости BSY (если устройство занято, то остальные разряды могут содержать недостоверную информацию), затем — бит ошибки, а в последнюю очередь — признаки готовности к приему команды DRDY или передаче данных DRQ. Несмотря на использование самых примитивных режимов, процедуры из листинга 6.7 являются достаточно эффективными при работе в однозадачном режиме DOS благодаря наличию у современных жестких дисков встроенной кэш-памяти большого объема.

Обратите внимание на необходимость защиты от зависания по длительности выполнения операции, которая должна быть встроена в циклы опроса регистра состояния. В данном примере использовалось с указанной целью значение состояния системного таймера, которое периодически (18 раз в секунду) записывается операционной системой в область данных BIOS (в 32-разрядное слово по абсолютному адресу 46Ch). Длительность ожидания сигнала готовности устройства, вообще говоря, зависит от типа этого устройства — современные дисководы могут автоматически переходить в спящее состояние, для возврата из которого в рабочий режим может потребоваться несколько секунд (для разгона мотора дисковода). В данном случае задано предельное значение длительности ожидания, равное 10 тикам системного таймера, что соответствует примерно 0,5 с.

Листинг 6.7. Процедуры для непосредственной работы с контроллером жестких дисков АТА

```
; Максимальное время ожидания завершения операции
; (в "тиках")
MaxHDDWaitTime equ 10

DATASEG
; Стандартные базовые адреса каналов 1 - 4
StandardHDDBases DW 1F0h, 170h, 1E8h, 168h
; Номер канала
ChannelNumber DW ?
; Базовый адрес группы портов контроллера HDD
HDDBasePortAddr DW ?
; Номер диска
HDDNumber DB ?
; Параметры ATA-команды
ATAFeatures     DB ? ;особенности
ATASectorCount  DB ? ;количество обрабатываемых секторов
ATASectorNumber DB ? ;номер начального сектора
ATACylinder     DW ? ;номер начального цилиндра
ATAHead         DB ? ;номер начальной головки
ATAAddressMode  DB ? ;режим адресации (0 - CHS, 1 - LBA)
ATACommand      DB ? ;код команды, подлежащей выполнению
; Код ошибки (0 - нет ошибок, 1 - превышен допустимый
; интервал ожидания, 2 - неверный код режима адресации,
; 3 - неверный номер канала, 4 - неверный номер диска,
; 5 - неверный номер головки, 6 - ошибка при выполнении
; команды)
HDDErrorCode DB ?
; Момент начала очередной операции с диском
HDDTime DD ?
; Адрес считываемого сектора в режиме LBA
SectorAddress DD 0
; Область памяти для хранения прочитанного сектора
SectorDataBuffer DB 512 DUP (?)
ENDS

CDDESEG
;************************************************
;* ЧТЕНИЕ СЕКТОРА ЖЕСТКОГО ДИСКА В РЕЖИМЕ LBA    *
;* Входные параметры передаются через глобальные *
;* переменные:                                   *
;* ChannelNumber - номер канала (1 - 4);         *
;* HDDNumber - номер диска на канале (0 или 1);  *
;* SectorAddress - номер считываемого сектора.   *
;* Сектор считывается в основной сегмент данных, *
;* в массив SectorDataBuffer.                    *
;************************************************
PROC ReadHDDSector NEAR
```

продолжение ↗

Листинг 6.7 (*продолжение*)

```
        pushad
        push    ES
; Задать режим LBA
        mov     [ATAAddressMode],1
; Послать команду чтения сектора (с повторами)
        mov     [ATAFeatures],0
        mov     [ATASectorCount],1
        mov     EAX,[SectorAddress]
        mov     [dword ptr ATASectorNumber],EAX
        mov     [ATACommand],20h
        call    SendCommandToHDD
        cmp     [HDDErrorCode],0
        jne     @@End    ;закончить, сохранив код ошибки
; Ожидать готовности данных HDD
        mov     AX,0
        mov     ES,AX
        mov     DX,[HDDBasePortAddr]
        add     DX,7     ;адрес регистра состояния
@@WaitCompleet:
        ; Проверить время выполнения команды
        mov     EAX,[ES:046Ch]
        sub     EAX,[HDDTime]
        cmp     EAX,MaxHDDWaitTime
        ja      @@Error1 ;ошибка тайм-аута
        ; Проверить готовность
        in      AL,DX
        test    AL,80h   ;состояние сигнала BSY
        jnz     @@WaitCoumpleet
        test    AL,08h   ;состояние сигнала DRQ
        jz      @@WaitCompleet
; Принять сектор
        mov     AX,[CS:MainDataSeg]
        mov     ES,AX
        mov     DI,offset SectorDataBuffer
        mov     DX,[HDDBasePortAddr] ;регистр данных
        mov     CX,256   ;число считываемых слов
        rep     insw     ;принять блок данных
; Сбросить признак ошибки
        mov     [HDDErrorCode],0
        jmp short @@End
; Записать номер ошибки
@@Error1:
        mov     [HDDErrorCode],1 ;ошибка тайм-аута
        jmp short @@End
@@End:  pop     ES
        popad
        ret
ENDP ReadHDDSector
```

```
;**************************************************
;*          ПОСЛАТЬ КОМАНДУ ЗАДАННОМУ ДИСКУ        *
;* Входные параметры передаются через глобальные   *
;* переменные:                                     *
;* ChannelNumber - номер канала (1 - 4);           *
;* HDDNumber - номер диска (0 или 1);              *
;* ATAFeatures - "особенности";                    *
;* ATASectorCount - количество секторов;           *
;* ATASectorNumber - номер начального сектора;     *
;* ATACylinder - номер начального цилиндра;        *
;* ATAHead - номер начальной головки;              *
;* ATAAddressMode - режим адресации (0-CHS, 1-LBA); *
;* ATACommand - код команды.                       *
;* После успешного выполнения функции:             *
;* в SectorDataBuffer - прочитанный сектор;        *
;* в HDDBasePortAddr - базовый адрес HDD;          *
;* в HDDTime - момент начала выполнения команды;   *
;* в HDDErrorCode - ноль.                          *
;* При возникновении ошибки в HDDErrorCode будет   *
;* возвращен код ошибки.                           *
;**************************************************
PROC SendCommandToHDD near
        pushad
        push    ES
; Запомнить время начала операции с диском
        ; Загрузить в ES сегмент данных BIDS
        mov     AX,0
        mov     ES,AX
        ; Запомнить текущее время
        mov     EAX,[ES:046Ch]
        mov     [HDDTime],EAX
; Проверить значение кода режима
        cmp     [ATAAddressMode],1
        ja      @@Error2
; Проверить корректность номера канала
        mov     BX,[ChannelNumber]
        cmp     BX,1
        jb      @@Error3
        cmp     BX,4
        ja      @@Error3
; Установить базовый адрес
        dec     BX
        shl     BX,1
        mov     AX,[BX+StandardHDDBases]
        mov     [HDDBasePortAddr],AX
; Запретить прерывания от контроллера HDD
        mov     DX,[HDDBasePortAddr]
        add     DX,206h
        mov     AL,1010b
        out     DX,AL
```

продолжение

Листинг 6.7 (*продолжение*)

```
; Ожидать "освобождения" HDD
        mov     DX,[HDDBasePortAddr]
        add     DX,7        ;адрес регистра состояния
@@WaitNotBSY:
        ; Проверить время выполнения команды
        mov     EAX,[ES:046Ch]
        sub     EAX,[HDDTime]
        cmp     EAX,MaxHDDWaitTime
        ja      @@Error1 ;ошибка тайм-аута
        ; Проверить состояние сигнала BSY
        in      AL,DX
        test    AL,80h      ;состояние сигнала BSY
        jnz     @@WaitNotBSY

; Ожидание готовности HDD к приему команды
        ; Выбрать нужный диск
        mov     DX,[HDDBasePortAddr]
        add     DX,6        ;адрес регистра головок
        mov     AL,[HDDNumber]
        cmp     AL,1        ;проверить номера диска
        ja      @@Error4
        shl     AL,4
        or      AL,10100000b
        out     DX,AL
        ; Ожидать, пока диск не будет готов
        inc     DX
@@WaitHDReady:
        ; Проверить время выполнения команды
        mov     EAX,[ES:046Ch]
        sub     EAX,[HDDTime]
        cmp     EAX,MaxHDDWaitTime
        ja      @@Error1 ;ошибка тайм-аута
        ; Проверить состояние BSY и DRDY
        in      AL,DX
        test    AL,80h      ;состояние сигнала BSY
        jnz     @@WaitHDReady
        test    AL,40h      ;состояние сигнала DRDY
        jz      @@WaitHDReady

; Загрузить команду в регистры контроллера
        mov     DX,[HDDBasePortAddr]
        inc     DX          ;регистр "особенностей"
        mov     AL,[ATAFeatures]
        out     DX,AL
        inc     DX          ;счетчик секторов
        mov     AL,[ATASectorCount]
        out     DX,AL
        inc     DX          ;регистр номера сектора
        mov     AL,[ATASectorNumber]
```

```
        out     DX,AL
        inc     DX       ;номер цилиндра (младший байт)
        mov     AX,[ATACylinder]
        out     DX,AL
        inc     DX       ;номер цилиндра (старший байт)
        mov     AL,AH
        out     DX,AL
        inc     DX       ;номер головки/номер диска
        mov     AL,[HDDNumber]
        shl     AL,4
        cmp     [ATAHead],0Fh  ;проверить номер головки
        ja      @@Error5
        or      AL,[ATAHead]
        or      AL,10100000b
        mov     AH,[ATAAddressMode]
        shl     AH,6
        or      AL,AH
        out     DX,AL
; Послать команду
        mov     AL,[ATACommand]
        inc     DX       ;регистр команд
        out     DX,AL
; Сбросить признак ошибки
        mov     [HDDErrorCode],0
        jmp short @@End

; Записать код ошибки
@@Error1:
        mov     [HDDErrorCode],1
        jmp short @@End
@@Error2:
        mov     [HDDErrorCode],2
        jmp short @@End
@@Error3:
        mov     [HDDErrorCode],3
        jmp short @@End
@@Error4:
        mov     [HDDErrorCode],4
        jmp short @@End
@@Error5:
        mov     [HDDErrorCode],5
        jmp short @@End
@@End:  pop     ES
        popad
        ret
ENDP SendCommandToHDD

;*************************************************
;*     ЧТЕНИЕ ИДЕНТИФИКАТОРА ЖЕСТКОГО ДИСКА      *
;* Входные параметры передаются через глобальные *
```

продолжение ⇗

Листинг 6.7 (*продолжение*)

```
;* переменные:                                    *
;* ChannelNumber - номер канала (1 - 4);          *
;* HDDNumber - номер диска на канале (0 или 1).   *
;* Идентификационный блок данных считывается      *
;* в массив SectorDataBuffer.                     *
;*************************************************
PROC ReadHDD_ID near
        pushad
        push    ES
; Задать режим CHS
        mov     [ATAAddressMode],0
; Послать команду идентификации устройства
        mov     [ATAFeatures],0
        mov     [ATAHead],0
        mov     [ATACommand],0ECh
        call    SendCommandToHDD
        cmp     [HDDErrorCode],0 ;проверить код ошибки
        jne     @@End  ;закончить, сохранив код ошибки
; Ожидать готовности данных HDD
        mov     AX,0
        mov     ES,AX
        mov     DX,[HDDBasePortAddr]
        add     DX,7      ;адрес регистра состояния
@@WaitCompleet:
        ; Проверить время выполнения команды
        mov     EAX,[ES:046Ch]
        sub     EAX,[HDDTime]
        cmp     EAX,MaxHDDWaitTime
        ja      @@Error1 ;ошибка тайм-аута
        ; Проверить готовность
        in      AL,DX
        test    AL,80h   ;состояние сигнала BSY
        jnz     @@WaitCompleet
        test    AL,1     ;состояние сигнала ERR
        jnz     @@Error6
        test    AL,08h   ;состояние сигнала DRQ
        jz      @@WaitCompleet
; Принять блок данных от контроллера
        mov     AX,[CS:MainDataSeg]
        mov     ES,AX
        mov     DI,offset SectorDataBuffer
        mov     DX,[HDDBasePortAddr] ;регистр данных
        mov     CX,256   ;число считываемых слов
        rep     insw     ;принять блок данных
        jmp short @@End
; Записать код ошибки
@@Error1:
        mov     [HDDErrorCode],1
        jmp short @@End
```

```
@@Error6:
        mov     [HDDErrorCode],6
@@End:  pop     ES
        popad
        ret
ENDP ReadHDD_ID
ENDS
```

Программа IdentifyDevices, приведенная в листинге 6.8, осуществляет поиск жестких дисков по каналам 1 (Primary) и 2 (Secondary) ATA-контроллера, считывает и отображает на экран параметры найденных устройств. Программа использует процедуры ввода-вывода общего назначения из глав «Работа с клавиатурой» и «Недокументированные возможности процессоров Intel 80x86», процедуры считывания данных с диска из листинга 6.7, а также две вспомогательные подпрограммы:

- процедура ShowHDD_ID отображает на экран содержимое некоторых полей 512-байтной структуры-идентификатора диска;
- процедура ClearPrevInfo очищает экран и выводит текстовые сообщения в верхней (заголовок) и нижней (надпись **Ждите**) строках экрана.

Листинг 6.8. Поиск подключенных жестких дисков по каналам 1 и 2

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Текстовые сообщения
Txt1 DB LIGHTCYAN,0,26,"ИДЕНТИФИКАЦИЯ ЖЕСТКИХ ДИСКОВ",0
     DB YELLOW,24,35,"Ждите ...",0
Txt2 DB 2,27,"На канале  найден диск  .",0
     DB 4,25,"Параметры обнаруженного диска:",0
     DB 6,22,"Общая информация:",0
     DB 7,12,"Число логических цилиндров:",0
     DB 8,14,"Число логических головок:",0
     DB 9,13,"Число логических секторов:",0
     DB 10,24,"Серийный номер:",0
     DB 11,32,"Модель:",0
     DB 12,8,"Макс. количество секторов за сеанс:",0
```

продолжение ⇨

Листинг 6.8 (*продолжение*)

```
     DB 13,27,"Возможности:",0
     DB 14,10,"Текущее число лог. цилиндров:",0
     DB 15,12,"Текущее число лог. головок:",0
     DB 16,11,"Текущее число лог. секторов:",0
     DB 17,13,"Текущая емкость секторов:",0
     DB 18,11,"Число секторов в режиме LBA:",0
     DB 19,13,"Поддерживаемые режимы DMA:",0
     DB 20,17,"Улучшенные режимы PIO:",0
Txt3 DB LIGHTGREEN
     DB 12,24,"Поиск завершен, больше нет дисков",0
AnyK DB YELLOW,24,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
        DB 400h DUP(?)
ENDS


CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC IdentifyDevices
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 2,Txt1

; Цикл опроса каналов
        mov     [ChannelNumber],1
; Опросить Master-диск
@@AskMaster:
        call    ClearPrevInfo
        mov     [HDDNumber],0
        call    ReadHDD_ID
        cmp     [HDDErrorCode],0
        jne     @@AskSlave
        call    ShowHDD_ID
; Опросить Slave-диск
@@AskSlave:
        call    ClearPrevInfo
```

```
        mov     [HDDNumber],1
        call    ReadHDD_ID
        cmp     [HDDErrorCode],0
        jne     @@NextCannel
        call    ShowHDD_ID
@@NextCannel:
        inc     [ChannelNumber]
        cmp     [ChannelNumber],3
        jb      @@AskMaster

@@End:  ; Выдать сообщение о завершении поиска
        call    ClearPrevInfo
        MShowColorText 2,Txt3
        call    GetChar
; Переустановить текстовый режим
        mov     ax,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP IdentifyDevices

;***********************************
;* РАСШИФРОВКА ИДЕНТИФИКАТОРА ДИСКА *
;***********************************
PROC ShowHDD_ID near
        pushad
; Установить зеленый цвет и черный фон
        mov     [TextColorAndBackground],LIGHTGREEN
        MShowText 17,Txt2
        MShowColorString AnyK
        ; Установить белый цвет и черный фон
        mov     [TextColorAndBackground],WHITE
; Вывести номер канала и номер диска
        MShowDecWord 2,37,[ChannelNumber]
        MShowDecByte 2,51,[HDDNumber]
; Вывести отдельные поля идентификатора
; (только неспецифические)
        ; Общая информация
        MShowBinWord 6,40,<[word ptr SectorDataBuffer]>
        ; Число логических цилиндров
        MShowDecWord 7,40,<[word ptr SectorDataBuffer+1*2]>
        ; Число логических головок
        MShowDecWord 8,40,<[word ptr SectorDataBuffer+3*2]>
        ; Число логических секторов
        MShowDecWord 9,40,<[word ptr SectorDataBuffer+6*2]>
        ; Серийный номер
        mov     AX,0B800h
        mov     ES,AX
        mov     SI,offset SectorDataBuffer
```

продолжение ⇒

Листинг 6.8 (*продолжение*)

```
        add     SI,10*2
        mov     DI,(10*80+40)*2
        mov     AH,[TextColorAndBackground]
        mov     CX,10
@@NextWord1:
        mov     DX,[SI]
        mov     AL,DH
        stosw
        mov     AL,DL
        stosw
        add     SI,2
        loop    @@NextWord1
        ; Номер модели
        mov     SI,offset SectorDataBuffer
        add     SI,27*2
        mov     DI,(11*80+40)*2
        mov     AH,[TextColorAndBackground]
        mov     CX,20
@@NextWord2:
        mov     DX,[SI]
        mov     AL,DH
        stosw
        mov     AL,DL
        stosw
        add     SI,2
        loop    @@NextWord2
        ; Макс. кол-во секторов за сеанс
        MShowDecByte 12,40,<[SectorDataBuffer+47*2]>
        ; Возможности
        MShowBinWord 13,40,<[word ptr SectorDataBuffer+49*2]>
        ; Значения слов 54-58 достоверны?
        test    [word ptr SectorDataBuffer+53*2],1
        jz      @@NotValid5458
        ; Текущее число логических цилиндров
        MShowDecWord 14,40,<[word ptr SectorDataBuffer+54*2]>
        ; Текущее число логических головок
        MShowDecWord 15,40,<[word ptr SectorDataBuffer+55*2]>
        ; Текущее число логических секторов
        MShowDecWord 16,40,<[word ptr SectorDataBuffer+56*2]>
        ; Текущая емкость секторов
        MShowDecDWord 17,40,<[dword ptr SectorDataBuffer+57*2]>
@@NotValid5458:
        ; Число секторов в режиме LBA
        MShowDecDWord 18,40,<[dword ptr SectorDataBuffer+60*2]>
        ; Поддерживаемые режимы DMA
        MShowBinByte 19,40,<[SectorDataBuffer+63*2]>
        ; Значения слов 64-70 достоверны?
        test    [word ptr SectorDataBuffer+53*2],10b
```

```
        jz      @@NotValid6470
        ; Поддерживаемые режимы PIO
        MShowBinByte 20,40,<[SectorDataBuffer+64*2]>
@@NotValid6470:
        ; Ожидать нажатия любой клавиши
        call    GetChar
        popad
        ret
ENDP ShowHDD_ID

;*************************************
;* ОЧИСТИТЬ ЭКРАН И ВЫВЕСТИ ЗАГОЛОВКИ *
;*************************************
PROC ClearPrevInfo near
        pushad
; Очистить экран
        call    ClearScreen
; Вывести текстовые сообщения на экран
        MShowColorText 2,Txt1
        popad
        ret
ENDP ClearPrevInfo
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить "математические" процедуры для перевода
; чисел из двоичного кода в десятичный
include "list2_05.inc"
; Подключить процедуру непосредственного считывания
; сектора с жесткого диска
include "list6_07.inc"

END
```

ПРИМЕЧАНИЕ

Для запуска программы lst_6_08.exe можно использовать любой AT-совместимый персональный компьютер с контроллером ATA.

В листинге 6.9 приведена программа ShowHDDSector, позволяющая осуществлять считывание (в режиме LBA) и просмотр секторов Master-диска Primary-канала (канала 1). Просмотр осуществляется так же, как и в листинге 6.6 — при помощи клавиш ↓ и ↑; для выхода из программы используется клавиша Esc. Программа использует процедуры ввода-вывода из главы 1 «Работа с клавиатурой» и главы 2 «Недокументированные возможности процессоров Intel 80x86», а также из листинга 6.7.

Листинг 6.9. Просмотр секторов ведущего жесткого диска канала 1 в режиме LBA

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Текстовые сообщения
Txt1 DB LIGHTCYAN,0,16,"Просмотр секторов "
     DB "жесткого диска в ASCII-кодах",0
     DB LIGHTGREEN,5,8,"Сектор N          :",0
     DB LIGHTCYAN,17,8,"Управляющие клавиши:",0
     DB YELLOW,24,27,"Нажмите управляющую клавишу",0
Txt2 DB 19,8,"Стрелка вниз - следующий сектор;",0
     DB 20,8,"Стрелка вверх - предыдущий сектор;",0
     DB 21,8,"Esc - выход.",0
Err1 DB 12,22,"Master-диск на канале 1 не обнаружен",0
Err2 DB 12,25,"Диск не поддерживает режим LBA",0
ENDS

SEGMENT sseg para stack 'STACK'
        DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC ShowHDDSector
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Опросить Master-диск канала 1
        mov     [ChannelNumber],1
        mov     [HDDNumber],0
        call    ReadHDD_ID
```

```
        ; Диск подключен?
        cmp     [HDDErrorCode],0
        jne     @@DiskNotFound
        ; Режим LBA поддерживается?
        cmp     [dword ptr SectorDataBuffer+60*2],0
        je      @@LBANotSupported
; Вывести текстовые сообщения на экран
        MShowColorText 4,Txt1
        ; Установить зеленый цвет и черный фон
        mov     [TextColorAndBackground],LIGHTGREEN
        MShowText 3,Txt2

; Инициализируем переменные
        mov     [HDDNumber],0
        mov     [dword ptr SectorAddress],0
        mov     AX,0BB00h
        mov     ES,AX
; ВНЕШНИЙ ЦИКЛ
@@ReadSector:
        MShowHexDword 5,17,[SectorAddress]
        call    ReadHDDSector
        mov     DI,7*160+8*2
        mov     SI,offset SectorDataBuffer
        ; Задать для символов светло-голубой
        ; цвет и синий фон
        mov     AH,LIGHTCYAN+BLUE*16
        mov     DX,8  ;счетчик строк
@@OutNextString:
        mov     CX,64 ;счетчик символов в строке
@@OutNextChar:
        lodsb
        stosw
        loop    @@OutNextChar
        add     DI,16*2
        dec     DX
        jnz     @@OutNextString

@@GetCommand:
        ; Ожидаем ввода следующей команды
        call    GetChar
        cmp     AL,0
        je      @@TestCommandByte
        call    Beep
        jmp short @@GetCommand

@@TestCommandByte:
        cmp     AH,B_Esc        ;"Выход"
        je      @@End
@@TestDn:
        cmp     AH,B_DN         ;"Стрелка вниз"
```

продолжение ⇒

Листинг 6.9 (*продолжение*)

```
        jne     @@TestUp
        ; Увеличить на 1 номер сектора
        inc     [dword ptr SectorAddress]
        jmp     @@ReadSector
@@TestUp:
        cmp     AH,B_UP         ;"Стрелка вверх"
        jne     @@CommandError
        cmp     [dword ptr SectorAddress],0
        je      @@CommandError
        ; Уменьшить на 1 номер сектора
        dec     [dword ptr SectorAddress]
        jmp     @@ReadSector
@@CommandError:
        call    Beep
        jmp     @@GetCommand

; Завершение работы программы
@@End:  ; Переустановить текстовый режим (очистить экран)
        mov     ax,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h

; Обработка ошибок
@@DiskNotFound:
        MFatalError Err1 ;диск не найден
@@LBANotSupported:
        MFatalError Err2 ;диск не поддерживает LBA
ENDP ShowHDDSector
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подклшчить процедуру непосредственного считывания
; сектора с жесткого диска
include "list6_07.inc"

END
```

ПРИМЕЧАНИЕ

Для запуска программы пригоден любой AT-совместимый персональный компьютер с жестким диском, установленным в качестве основного на канале 1 и поддерживающим режим LBA. Чтобы проверить наличие Master-диска на Primary-канале и способность этого диска работать в режиме LBA, можно использовать программу поиска дисков из листинга 6.8.

Приведенная в листинге 6.10 программа SearchLogicalDisks осуществляет поиск логических дисков на Master-диске Primary-канала. Работа с диском ведется в режиме LBA. Программа выводит на экран таблицу разделов и расшифровку полей загрузочного сектора каждого найденного логического диска при помощи процедур ShowPartitionTable и DecodeBootSector. Для отображения на экран текстовых полей используется вспомогательная процедура ShowASCIIField. Программа lst_6_10.exe предъявляет к аппаратуре те же самые требования, что и программа из листинга 6.9.

Листинг 6.10. Просмотр в режиме LBA параметров логических дисков на ведущем жестком диске канала 1

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Текстовые сообщения
Txt00 DB YELLOW,24,29,"Нажмите любую клавишу",0
Txt01 DB YELLOW,0,30
      DB "MASTER-ДИСК КАНАЛА 1",0
      DB YELLOW,1,22
      DB "ТАБЛИЦА РАЗДЕЛОВ ЛОГИЧЕСКОГО ДИСКА",0
Txt02 DB 8,1,"N",0
      DB 8,4,"Тип",0
      DB 8,9,"Призн.",0,9,9,"акт.",0
      DB 8,16,"  Начало раздела",0
      DB 9,17,"Гол.",0,9,24,"Цил.",0,9,31,"Сект.",0
      DB 8,37,"  Конец раздела",0
      DB 9,38,"Гол.",0,9,45,"Цил.",0,9,52,"Сект.",0
      DB 8,58,"Номер нач.",0
      DB 9,58,"сектора",0
      DB 8,70,"Размер,",0
      DB 9,70,"секторов",0
Txt11 DB YELLOW,0,19
      DB "ЗАГРУЗОЧНАЯ ЗАПИСЬ ЛОГИЧЕСКОГО ДИСКА N",0
Txt12 DB 2,12,"Идентификатор изготовителя:",0
      DB 3,22,"Байтов на сектор:",0
      DB 4,19,"Секторов в кластере:",0
      DB 5,14,"Число резервных секторов:",0
```

продолжение ⇒

Листинг 6.10 (*продолжение*)

```
      DB 6,23,"Число копий FAT:",0
      DB 7,6,"Дескрипторов в корневом каталоге:",0
      DB 8,14,"Число секторов в разделе:",0
      DB 9,15,"Тип носителя информации:",0
      DB 10,1,"Число секторов, занимаемых копией FAT:",0
      DB 11,13,"Число секторов на дорожке:",0
      DB 12,1,"Количество рабочих поверхностей диска:",0
      DB 13,16,"Число скрытых секторов:",0
Txt13 DB 14,23,"Номер дисковода:",0
      DB 15,15,"Номер логического диска:",0
      DB 16,27,"Метка диска:",0
      DB 17,9,"Аббревиатура файловой системы:",0
Txt14 DB 14,20,"Номер активной FAT:",0
      DB 15,22,"Номер версии FAT:",0
      DB 16,0
      DB "Номер нач. кластера корневого каталога:",0
      DB 17,8,"Номер сектора структуры FSINFO:",0
      DB 18,4,"Номер сектора резервной копии Boot:",0
      DB 19,23,"Номер дисковода:",0
      DB 20,15,"Номер логического диска:",0
      DB 21,27,"Метка диска:",0
      DB 22,9,"Аббревиатура файловой системы:",0
Err1  DB 12,22,"Master-диск на канале 1 не обнаружен",0
Err2  DB 12,25,"Диск не поддерживает режим LBA",0
Err3  DB 12,25,"Основной раздел DOS не найден",0
; Номер раздела
PartitionNumber DB ?
; Начальный сектор основного раздела DOS
PriDOS_StartSector DD ?
; Начальный сектор расширенного раздела DOS
ExtDOS_StartSector DD ?
; Начальный сектор текущего логического диска
CurrentDrive_StartSector DD ?
; Линейный адрес загрузочного сектора
BootSector DD ?
; Номер логического диска
LogicalDriveNumber DB ?
; Флаг присутствия в системе следующего диска
NextDrivePresent DB ?
ENDS

SEGMENT sseg para stack 'STACK'
        DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
```

```
PROC SearchLogicalDisks
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Опросить Master-диск канала 1
        mov     [ChannelNumber],1
        mov     [HDDNumber],0
        call    ReadHDD_ID
        ; Диск подключен?
        cmp     [HDDErrorCode],0
        jne     @@DiskNotFound
        ; Режим LBA поддерживается?
        cmp     [dword ptr SectorDataBuffer+60*2],0
        je      @@LBANotSupported

; Обнулить номер логического диска
        mov     [LogicalDriveNumber],0
; Прочитать MBR диска
        mov     [CurrentDrive_StartSector],0
        mov     [dword ptr SectorAddress],0
        call    ReadHDDSector
; Отобразить таблицу разделов
        call    ShowPartitionTable

; Найти и записать номера начальных секторов
; основного и расширенного разделов DOS
        mov     [NextDrivePresent],0
        mov     SI,offset SectorDataBuffer
        add     SI,1BEh      ;смещение первой записи
        mov     AL,[SI+4]    ;извлечь тип раздела
        ; Проверить код основного раздела
        cmp     AL,01h
        je      @@PrimPartFound
        cmp     AL,04h
        je      @@PrimPartFound
        cmp     AL,06h
        je      @@PrimPartFound
        cmp     AL,0Bh
        je      @@PrimPartFound
        cmp     AL,0Eh
        jne     @@PrimPartNotFound
@@PrimPartFound:
        ; Найден основной раздел, сохранить
```

продолжение

Листинг 6.10 (*продолжение*)

```
        ; адрес его начального сектора
        mov     EAX,[SI+8] ;извлечь адрес сектора
        mov     [PriDOS_StartSector],EAX
        add     SI,10h      ;смещение второй записи
        mov     AL,[SI+4]  ;извлечь тип раздела
        ; Проверить код расширенного раздела
        cmp     AL,05h
        je      @@ExtPartFound
        cmp     AL,0Ch
        je      @@ExtPartFound
        cmp     AL,0Fh
        jne     @@NextDriveNotPresent0
@@ExtPartFound:
        ; Имеется расширенный раздел
        mov     EAX,[SI+8] ;извлечь адрес сектора
        mov     [ExtDOS_StartSector],EAX
        mov     [NextDrivePresent],1
@@NextDriveNotPresent0:

; Прочитать BOOT-сектор основного раздела
        mov     EAX,[PriDOS_StartSector]
        mov     [SectorAddress],EAX
        call    ReadHDDSector
; Отобразить данные BOOT-сектора основного раздела
        call    DecodeBootSector

        ; Имеется следующий диск?
        cmp     [NextDrivePresent],0
        je      @@End
        mov     EAX,[ExtDOS_StartSector]
        mov     [CurrentDrive_StartSector],EAX

; ЦИКЛ ОПРОСА ЛОГИЧЕСКИХ ДИСКОВ РАСШИРЕННОГО РАЗДЕЛА
@@ReadSMBR:
        inc     [LogicalDriveNumber]
        ; Прочитать очередной SMBR
        mov     EAX,[CurrentDrive_StartSector]
        mov     [BootSector],EAX
        mov     [SectorAddress],EAX
        call    ReadHDDSector
        ; Отобразить таблицу разделов
        call    ShowPartitionTable

; Найти и записать номера начальных секторов
        mov     [NextDrivePresent],0
        mov     SI,offset SectorDataBuffer
        add     SI,1BEh     ;смещение первой записи
        mov     EAX,[SI+8]
        add     [BootSector],EAX
```

```
        add     SI,10h     ;смещение второй записи
        mov     EAX,[SI+8]
        cmp     EAX,0      ;следующий диск присутствует?
        je      @@NextDriveNotPresent1
        ; Вычислить адрес SMBR следующего диска
        add     EAX,[ExtDOS_StartSector]
        mov     [CurrentDrive_StartSector],EAX
        ; Установить признак наличия следующего диска
        mov     [NextDrivePresent],1
@@NextDriveNotPresent1:

; Прочитать BOOT-сектор логического диска
        mov     EAX,[BootSector]
        mov     [SectorAddress],EAX
        call    ReadHDDSector
        ; Отобразить данные BOOT-сектора
        call    DecodeBootSector
; Имеется следующий диск?
        cmp     [NextDrivePresent],0
        je      @@End
        jmp     @@ReadSMBR

; Завершение работы программы
@@End:  ; Переустановить текстовый режим
        mov     ax,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h

; Обработка ошибок
@@DiskNotFound:
        MFatalError Err1 ;диск не найден
@@LBANotSupported:
        MFatalError Err2 ;диск не поддерживает LBA
@@PrimPartNotFound:
        MFatalError Err3 ;не найден основной раздел
ENDP SearchLogicalDisks

;****************************************
;* ПРОЧИТАТЬ НУЛЕВОЙ СЕКТОР И ОТОБРАЗИТЬ *
;*       НА ЭКРАН ТАБЛИЦУ РАЗДЕЛОВ       *
;****************************************
PROC ShowPartitionTable near
        pushad
        push    ES
; Очистить экран
        call    ClearScreen

; Вывести текстовые сообщения на экран
```

продолжение

Листинг 6.10 (*продолжение*)

```
        MShowColorString Txt00
        MShowColorText 2,Txt01
        mov     [TextColorAndBackground],YELLOW
        MShowHexByte 1,57,[LogicalDriveNumber]
; ЦИКЛ РИСОВАНИЯ ТАБЛИЦЫ
        mov     AX,0B800h     ;Настроить ES для прямого
        mov     ES,AX         ;вывода на экран
        mov     DI,B*B0*2     ;Начать вывод с B-й строки
        mov     AH,LIGHTBLUE  ;Чертить синим цветом
        mov     AL,0B3h       ;Задать символ-разделитель
        mov     DX,7          ;Задать общее число строк
; Отобразить символы-разделители колонок таблицы
@@ac0:  push    DI
        add     DI,3*2
        mov     [ES:DI],AX
        add     DI,5*2
        mov     CX,B
@@ac1:  mov     [ES:DI],AX
        add     DI,7*2
        loop    @@ac1
        add     DI,4*2
        mov     [ES:DI],AX
        pop     DI
        add     DI,80*2
        dec     DX
        jnz     @@ac0

; Установить зеленый цвет и черный фон
        mov     [TextColorAndBackground],LIGHTGREEN
; Отобразить шапку таблицы
        MShowText 16,Txt02

; ЦИКЛ ВЫВОДА ДАННЫХ
; Установить белый цвет и черный фон
        mov     [TextColorAndBackground],WHITE
; Отобразить загрузочный сектор на экран
        mov     [PartitionNumber],1
        mov     SI,offset SectorDataBuffer
        ; Прибавить смещение первой записи
        ; от начала сектора
        add     SI,1BEh
        ; Задать начальную строку для вывода данных
        mov     [ScreenString],11

@@ShS0: ; Отобразить порядковый номер раздела
        mov     AL,[PartitionNumber]
        mov     [ScreenColumn],0
        call    ShowHexByte
```

```
; Отобразить код типа раздела
lodsb
mov     [ScreenColumn],11
call    ShowHexByte

; Отобразить номер начальной поверхности раздела
lodsb
mov     [ScreenColumn],18
call    ShowHexByte

; Отобразить номер начального цилиндра
; и номер начального сектора раздела
lodsw
mov     BL,AL
; Вычислить и отобразить номер цилиндра
shr     AL,6
xchg    AL,AH
mov     [ScreenColumn],24
call    ShowHexWord
; Вычислить и отобразить номер сектора
mov     AL,BL
and     AL,0111111b
mov     [ScreenColumn],32
call    ShowHexByte

; Отобразить код признака активного раздела
mov     [ScreenColumn],5
lodsb
call    ShowHexByte

; Отобразить номер конечной поверхности раздела
mov     [ScreenColumn],39
lodsb
call    ShowHexByte

; Отобразить номер конечного цилиндра
; и номер конечного сектора раздела
lodsw
mov     BL,AL
; Вычислить и отобразить номер цилиндра
shr     AL,6
xchg    AL,AH
mov     [ScreenColumn],45
call    ShowHexWord
; Вычислить и отобразить номер сектора
mov     AL,BL
and     AL,0111111b
mov     [ScreenColumn],53
call    ShowHexByte
```

продолжение ⇗

Листинг 6.10 (*продолжение*)

```
        ; Отобразить абсолютный номер начального сектора
        mov     [ScreenColumn],59
        mov     EAX,[SI]
        add     SI,4
        call    ShowHexDWord

        ; Отобразить размер раздела в секторах
        mov     [ScreenColumn],70
        mov     EAX,[SI]
        add     SI,4
        call    ShowHexDWord

        inc     [ScreenString]
        inc     [PartitionNumber]
        cmp     [PartitionNumber],5
        jb      @@ShS0

        ; Ожидаем нажатия клавиши
        call    GetChar
        pop     ES
        popad
        ret
ENDP ShowPartitionTable

;*********************************************
;* ОТОБРАЗИТЬ ИНФОРМАЦИЮ ЗАГРУЗОЧНОГО СЕКТОРА *
;*********************************************
PROC DecodeBootSector near
        pushad
        push    ES
; Очистить экран
        call    ClearScreen
; Вывести текстовые сообщения на экран
        MShowColorString Txt00
        MShowColorString Txt11
; Отобразить порядковый номер логического диска
        mov     [TextColorAndBackground],YELLOW
        MShowHexByte 0,58,[LogicalDriveNumber]

; Установить зеленый цвет и черный фон
        mov     [TextColorAndBackground],LIGHTGREEN
; Отобразить названия общих полей
        MShowText 12,Txt12

; Настроить ES для прямого вывода на экран
        mov     AX,0B800h
        mov     ES,AX
; Установить указатель на область данных
        mov     SI,offset SectorDataBuffer
```

```
; Установить белый цвет и черный фон
        mov     [TextColorAndBackground],WHITE

; ОБЩИЙ УЧАСТОК BOOT-СЕКТОРА
; Отобразить идентификатор OEM с 40-й позиции 2-й строки
        MShowASCIIField 2,40,3,8
; Отобразить число байтов на сектор
        MShowHexWord    3,40,[SI+0Bh]
; Отобразить число секторов в кластере
        MShowHexByte    4,40,[SI+0Dh]
; Отобразить число резервных секторов
        MShowHexWord    5,40,[SI+0Eh]
; Отобразить число копий FAT в разделе
        MShowHexByte    6,40,[SI+10h]
; Отобразить число дескрипторов в корневом каталоге
        MShowHexWord    7,40,[SI+11h]
; Отобразить число секторов в разделе
        cmp     [word ptr SI+13h],0
        je      @@UseTotSec32 ;использовать BPB_TotSec32
        MShowHexWord    8,40,[SI+13h]
@@UseTotSec32:
; Отобразить тип носителя информации
        MShowHexByte    9,40,[SI+15h]
; Отобразить число секторов, занимаемых копией FAT
        cmp     [word ptr SI+16h],0
        je      @@UseFATSz32 ;использовать BPB_FATSz32
        MShowHexWord   10,40,[SI+16h]
@@UseFATSz32:
; Отобразить число секторов на дорожке
        MShowHexWord   11,40,[SI+1Bh]
; Отобразить количество рабочих поверхностей диска
        MShowHexWord   12,40,[SI+1Ah]
; Отобразить число скрытых секторов
        MShowHexDWord  13,40,[SI+1Ch]
; Отобразить число секторов в разделе
        cmp     [dword ptr SI+20h],0
        je      @@UseTotSec16 ;использовать BPB_TotSec16
        MShowHexDWord   8,40,[SI+20h]
@@UseTotSec16:

; Если размер корневого сектора равен нулю - FAT32
        cmp     [word ptr SI+11h],0
        je      @@FAT32
; УЧАСТОК BOOT-СЕКТОРА, СПЕЦИФИЧЕСКИЙ ДЛЯ FAT12 И FAT16
; Установить зеленый цвет и черный фон
        mov     [TextColorAndBackground],LIGHTGREEN
; Отобразить названия полей, специфических
; для FAT12 и FAT16
        MShowText 4,Txt13
; Установить белый цвет и черный фон
        mov     [TextColorAndBackground],WHITE
```


продолжение ⇨

Листинг 6.10 (*продолжение*)

```
; Установить указатель на область данных
        mov     SI,offset SectorDataBuffer
; Отобразить номер дисковода
        MShowHexByte    14,40,[SI+24h]
; Отобразить номер логического диска
        MShowHexDWord   15,40,[SI+27h]
; Отобразить метку диска
        MShowASCIIField 16,40,2Bh,11
; Отобразить аббревиатуру файловой системы
        MShowASCIIField 17,40,36h,8
        jmp     @@End

@@FAT32:
; УЧАСТОК BOOT-СЕКТОРА, СПЕЦИФИЧЕСКИЙ ДЛЯ FAT32
; Установить зеленый цвет и черный фон
        mov     [TextColorAndBackground],LIGHTGREEN
; Отобразить названия полей, специфических FAT32
        MShowText 9,Txt14
; Установить белый цвет и черный фон
        mov     [TextColorAndBackground],WHITE
; Установить указатель на область данных
        mov     SI,offset SectorDataBuffer
; Отобразить число секторов, занимаемых копией FAT32
        MShowHexDWord   10,40,[SI+24h]
; Отобразить номер активной FAT
        MShowHexWord    14,40,[SI+28h]
; Отобразить код версии FAT
        MShowHexWord    15,40,[SI+2Ah]
; Отобразить номер начального кластера корневого
; каталога
        MShowHexDWord   16,40,[SI+2Ch]
; Отобразить номер сектора структуры FSINFO
        MShowHexWord    17,40,[SI+30h]
; Отобразить номер сектора резервной копии Boot-сектора
        MShowHexWord    18,40,[SI+32h]
; Отобразить номер дисковода
        MShowHexByte    19,40,[SI+40h]
; Отобразить номер логического диска
        MShowHexDWord   20,40,[SI+43h]
; Отобразить метку диска
        MShowASCIIField 21,40,47h,11
; Отобразить аббревиатуру файловой системы
        MShowASCIIField 22,40,52h,8

@@End:  call    GetChar
        pop     ES
        popad
        ret
ENDP DecodeBootSector
```

```
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуру непосредственного считывания
; сектора с жесткого диска
include "list6_07.inc"

END
```

Листинг 6.11 содержит программу SearchAutoexecBat, которая осуществляет поиск файла с именем autoexec.bat в корневой папке первого логического диска Master-диска Primary-канала (то есть диска, который обычно служит для загрузки операционной системы и обязательно содержит подобный файл), а затем отображает содержимое файла на экран. Процесс поиска программа иллюстрирует, отображая на экране содержимое начальных участков корневой папки и FAT логического диска.

Листинг 6.11. Поиск файла AUTOEXEC.BAT в корневом каталоге ведущего диска канала 1, считывание и отображение его на экране

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мненонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

; Размер корневого каталога в секторах
RootDirSize equ 32
; Адрес области дополнительной памяти, выделенной
; для хранения FAT (размер области 128 Кб)
FATAreaAddress equ 110000h
; Адрес области дополнительной памяти, выделенной для
; хранения корневого каталога (размер области 16 Кб)
RootDirAddress equ 130000h
; Адрес области дополнительной памяти, выделенной для
; хранения файла
FileBufferAddress equ 134000h

DATASEG
; Текстовые сообщения
Txt00 DB YELLOW,24,29,"Нажмите любую клавишу",0
```

продолжение ⇨

Листинг 6.11 (*продолжение*)

```
Txt20 DB YELLOW,0,25,"Начальная область первой FAT:",0
Txt30 DB YELLOW,0,23
      DB "Список файлов корневого каталога:",0
Txt40 DB YELLOW,0,16,"Просмотр начального "
      DB "участка файла AUTOEXEC.BAT:",0
Err1  DB 12,22,"Master-диск на канале 1 не обнаружен",0
Err2  DB 12,25,"Диск не поддерживает режим LBA",0
; Номер раздела
PartitionNumber    DB ?
; Начальный сектор основного раздела DOS
PriDOS_StartSector DD ?
; Линейный адрес загрузочного сектора
BootSector         DD ?
; Флаг присутствия в системе следующего диска
NextDrivePresent   DB ?
; Число секторов в кластере
SectorsInCluster   DW ?
; Число резервных секторов
RSects             DD ?
; Число таблиц FAT на диске
FATsOnDisk         DW ?
; Размер одной таблицы FAT в секторах
FATSize            DW ?
; Число скрытых секторов
HiddenSectors      DD ?
; Абсолютный номер начального сектора FAT
FATStartSect       DD ?
; Абсолютный номер начального сектора корневого каталога
RootDirStartSect   DD ?
; Начальный кластер файла
FileStartCluster   DW ?
; Абсолютный номер начального сектора файла
FileStartSect      DD ?
; Длина файла в байтах
FileSize           DD ?
ENDS

SEGMENT sseg para stack 'STACK'
        DB 400h DUP(?)
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC SearchAutoexecBat
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
```

```
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Установить режим прямой адресации памяти
        call    Initialization
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Опросить Master-диск канала 1
        mov     [ChannelNumber],1
        mov     [HDDNumber],0
        call    ReadHDD_ID
        ; Диск подключен?
        cmp     [HDDErrorCode],0
        jne     @@DiskNotFound
        ; Режим LBA поддерживается?
        cmp     [dword ptr SectorDataBuffer+60*2],0
        je      @@LBANotSupported

; Прочитать загрузочный сектор диска 0
        mov     [dword ptr SectorAddress],0
        call    ReadHDDSector
; Записать номер начального сектора основного раздела
        mov     SI,offset SectorDataBuffer
        add     SI,1BEh         ;смещение первой записи
        mov     EAX,[SI+8]
        mov     [PriDOS_StartSector],EAX

; Прочитать BOOT-сектор основного раздела диска 0
        mov     EAX,[PriDOS_StartSector]
        mov     [SectorAddress],EAX
        call    ReadHDDSector
; Записать основные параметры для расчетов
        mov     SI, offset SectorDataBuffer
        ; Число секторов в кластере
        movzx   AX,[byte ptr SI+0Dh]
        mov     [SectorsInCluster],AX
        ; Число резервных секторов
        movzx   EAX,[word ptr SI+0Eh]
        mov     [RSects],EAX
        ; Число копий FAT в разделе
        movzx   AX,[byte ptr SI+10h]
        mov     [FATsOnDisk],AX
        ; Размер одной таблицы FAT в секторах
        mov     AX,[SI+16h]
        mov     [FATSize],AX
        ; Число скрытых секторов
        mov     EAX,[SI+1Ch]
        mov     [HiddenSectors],EAX
```

продолжение ⇒

Листинг 6.11 (*продолжение*)

```
; ПРОЧИТАТЬ И ПОКАЗАТЬ FAT ОСНОВНОГО РАЗДЕЛА ДИСКА 0
; Очистить экран и вывести текстовые сообщения
        call    ClearScreen
        MShowColorString Txt20
        MShowColorString Txt00
; Скопировать FAT в дополнительную память
        ; Вычислить начальный сектор FAT
        mov     EAX,[PriDOS_StartSector]
        add     EAX,[RSects]
        mov     [FATStartSect],EAX
        mov     [SectorAddress],EAX
        ; Прочитать FAT
        mov     EDI,FATAreaAddress
        mov     DX,[FATSize] ;счетчик секторов FAT
@@NextFATSector:
        ; Прочитать очередной сектор FAT
        call    ReadHDDSector
        inc     [dword ptr SectorAddress]
        ; Переписать сектор FAT в дополнительную память
        mov     SI, offset SectorDataBuffer
        mov     CX,512
@@NextFATSectByte:
        lodsb
        mov     [GS:EDI],AL
        inc     EDI
        loop    @@NextFATSectByte
        dec     DX
        jnz     @@NextFATSector
; Показать на экране начальный участок FAT
        mov     [ScreenString],2
        mov     [ScreenColumn],0
        mov     EDI,FATAreaAddress
        mov     CX,256
@@FAT:  mov     AX,[GS:EDI]
        call    ShowHexWord
        inc     [ScreenColumn]
        add     EDI,2
        loop    @@FAT
; Ожидать нажатия любой клавиши
        call    GetChar

; ПРОЧИТАТЬ И ПОКАЗАТЬ КОРНЕВОЙ КАТАЛОГ
; ОСНОВНОГО РАЗДЕЛА ДИСКА 0
; Очистить экран и вывести текстовые сообщения
        call    ClearScreen
        MShowColorString Txt30
        MShowColorString Txt00
; Скопировать корневой каталог в дополнительную память
        ; Вычислить начальный сектор корневого каталога
```

```
        movzx   EAX,[FATSize]
        movzx   EDX,[FATsOnDisk]
        mul     EDX
        add     EAX,[FATStartSect]
        mov     [RootDirStartSect],EAX
        mov     [SectorAddress],EAX
        ; Прочесть корневой каталог
        mov     EDI,RootDirAddress
        mov     DX,RootDirSize ;счетчик секторов
@@NextRootSector:
        ; Прочитать очередной сектор каталога
        call    ReadHDDSector
        inc     [dword ptr SectorAddress]
        ; Переписать сектор в дополнительную память
        mov     SI, offset SectorDataBuffer
        mov     CX,512
@@NextRootByte:
        lodsb
        mov     [GS:EDI],AL
        inc     EDI
        loop    @@NextRootByte
        dec     DX
        jnz     @@NextRootSector
; Показать на экране начальный участок каталога
        mov     AX,0BB00h
        mov     ES,AX
        mov     DI,160*2
        mov     ESI,RootDirAddress
        mov     AH,WHITE
        mov     DX,100 ;счетчик записей
@@NextFileName:
        mov     CX,11
@@NextNameByte:
        mov     AL,[GS:ESI]
        stosw
        inc     [ScreenColumn]
        inc     ESI
        loop    @@NextNameByte
        add     DI,(16-11)*2
        add     ESI,32-11
        dec     DX
        jnz     @@NextFileName
; Ожидать нажатия любой клавиши
        call    GetChar

; НАЙТИ И ВЫВЕСТИ НА ЭКРАН ФАЙЛ AUTOEXEC.BAT
; Очистить экран и вывести текстовые сообщения
        call    ClearScreen
        MShowColorString Txt40
        MShowColorString Txt00
; Найти в корневом каталоге файл AUTOEXEC.BAT
```

продолжение ⇒

Листинг 6.11 (*продолжение*)

```
        mov     ESI,RootDirAddress
        mov     CX,512 ;число элементов в каталоге
@@NextElement:
        cmp     [dword ptr GS:ESI],'OTUA'
        jne     @@NotAutoexec
        cmp     [dword ptr GS:ESI+4],'CEXE'
        jne     @@NotAutoexec
        cmp     [word ptr GS:ESI+8],'AB'
        jne     @@NotAutoexec
        cmp     [byte ptr GS:ESI+10],'T'
        je      @@AutoexecFileFound
@@NotAutoexec:
        add     ESI,32 ;прибавить размер элемента
        loop    @@NextElement
        ; Ошибка - файл AUTOEXEC.BAT не найден

; Показать начальный участок файла AUTDEXEC.BAT
@@AutoexecFileFound:
        ; Определить начальный кластер файла
        mov     AX,[GS:ESI+1Ah]
        mov     [FileStartCluster],AX
        ; Определить длину файла в байтах
        mov     EAX,[GS:ESI+1Ch]
        mov     [FileSize],EAX
        ; Вычислить абсолютный номер сектора
        movzx   EAX,[FileStartCluster]
        sub     EAX,2
        movzx   EDX,[SectorsInCluster]
        mul     EDX
        add     EAX,[RootDirStartSect]
        add     EAX,RootDirSize
        mov     [FileStartSect],EAX
        mov     [SectorAddress],EAX
; Прочитать кластер
        mov     EDI,FileBufferAddress
        mov     DX,[SectorsInCluster]
@@NextFileSector:
        ; Прочитать очередной сектор файла
        call    ReadHDDSector
        inc     [dword ptr SectorAddress]
        ; Переписать сектор в дополнительную память
        mov     SI, offset SectorDataBuffer
        mov   . CX,512
@@NextFileByte:
        lodsb
        mov     [GS:EDI],AL
        inc     EDI
        loop    @@NextFileByte
        dec     DX
```

```
        jnz     @@NextFileSector
; Показать прочитанный файл
        mov     AX,0B800h
        mov     ES,AX
        mov     DI,160*2 ;вывод начать со 2-й строки
        mov     ESI,FileBufferAddress ;смещение данных
        mov     ECX,[FileSize]        ;размер файла
        mov     DX,0     ;счетчик строк экрана
        mov     BX,0     ;счетчик колонок экрана
        mov     AH,LIGHTGREEN
@@Out1: ; Загрузить очередной символ строки в AL
        mov     AL,[GS:ESI]
        inc     ESI
        ; Проверка на символ конца строки
        cmp     AL,0Dh
        jz      @@Out2
        ; Проверка на символ перевода строки
        ; (игнорировать его)
        cmp     AL,0Ah
        jz      @@Out3
        ; Вывести символ на экран
        stosw
        inc     BX
        cmp     BX,80
        jb      @@Out3
@@Out2: add     DI,160
        sub     DI,BX
        sub     DI,BX
        mov     BX,0
        inc     DX
        cmp     DX,22
        ja      @@OutEnd
@@Out3: dec     ECX
        jnz     @@Out1
@@OutEnd:
; Ожидать нажатия любой клавиши
        call    GetChar
; Переустановить текстовый режим
        mov     ax,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h

; Обработка ошибок
@@DiskNotFound:
        MFatalError Err1 ;диск не найден
@@LBANotSupported:
        MFatalError Err2 ;диск не поддерживает LBA
ENDP SearchAutoexecBat
ENDS
```

продолжение ⇨

Листинг 6.11 (*продолжение*)

```
; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить процедуру непосредственного считывания
; сектора с жесткого диска
include "list6_07.inc"

END
```

ПРИМЕЧАНИЕ

В приведенных примерах Master-диск Primary-раздела был выбран исходя из того, что в персональных компьютерах традиционно жесткий диск подключается именно таким образом (в старых моделях компьютеров такое подключение обеспечивало максимальную скорость передачи данных, а затем это стало фактическим стандартом, вошло в привычку).

Особенности реализации режима DMA на системных платах с шиной PCI

Режим прямого доступа к памяти (Direct Memory Access, сокращенно DMA) обеспечивает возможность обмена данными между периферийными устройствами и оперативной памятью компьютера без непосредственного участия процессора. В первую очередь он необходим в том случае, если используемые периферийные устройства имеют небольшие встроенные буферы памяти (такие устройства нуждаются в немедленном обслуживании и не могут ждать, пока процессор освободится от выполнения задач с более высоким приоритетом). Кроме того, в некоторых случаях режим DMA обеспечивает более высокую скорость передачи данных, чем режим PIO.

К сожалению, у современных персональных компьютеров прямой доступ к памяти реализован довольно заумным образом. Проблема DMA имеет исторические корни: при разработке архитектуры компьютеров с дешевыми 8-разрядными процессорами режим прямого доступа пытались вообще исключить из системы с целью снижения ее себестоимости. При переходе на микропроцессоры с 16-разрядной, а затем 32-разрядной архитектурой возникла потребность в режиме DMA, и инженерам пришлось использовать разнообразные ухищрения, чтобы не потерять совместимость со старым программным обеспечением.

Работа с жестким диском на персональных компьютерах, совместимых с IBM AT, изначально осуществлялась через процессор, то есть только в режиме PIO, так как канал DMA шины ISA вообще функционирует с недостаточно высокой скоростью, а для работы с диском был зарезервирован 8-разрядный канал. После того, как контроллер дисков был интегрирован в системную плату и подсоединен к шине PCI, появилась возможность использования канала DMA этой шины при работе с дисками. PCI DMA обеспечивает некоторый выигрыш по скорости по сравнению с PIO не только за счет собственно прямого доступа к памяти, но и за счет большей разрядности передаваемого слова (в режиме PIO — 16 разрядов, в режиме PCI DMA — 32 разряда).

Контроллер PCI IDE может работать в одном из двух режимов — в режиме совместимости или в режиме чистой PCI. В режиме совместимости размещение регистров и распределение линий прерывания полностью совпадает с тем, которое принято для шины ISA (см. табл. 6.27 и 6.28).

В режиме PCI имеется возможность произвольным образом задавать адреса групп регистров каналов IDE и номера выделенных им прерываний. При этом местоположение регистров хранится в конфигурационном пространстве PCI в 32-разрядных регистрах со следующими смещениями:

- смещение 10h — адрес блока регистров команды первичного канала;
- смещение 14h — адрес блока регистров контроля первичного канала;
- смещение 18h — адрес блока регистров команды вторичного канала;
- смещение 1Ch — адрес блока регистров контроля вторичного канала.

При работе в режиме чистой PCI обращаться к регистрам контроллера IDE можно только по адресам, заданным в регистрах конфигурационного пространства.

Работать с контроллером PCI IDE можно при помощи функций PCI BIOS, описанных в главе 3 «Особенности работы с устройствами, подключенными к шине PCI». Чтобы получить доступ к конфигурационному пространству контроллера PCI IDE, нужно определить его координаты на шине PCI, а сделать это можно либо по коду изделия и коду изготовителя, либо по коду класса контроллера. Первый способ пригоден только в том случае, если программное обеспечение

ориентировано на какой-либо чипсет определенного изготовителя. Второй способ более универсален, но требует перебора нескольких возможных вариантов значения младшего байта (байта интерфейса) кода класса (оба старших байта кода содержат значения 01h).

Разряды байта интерфейса кода класса (см. рис. 3.4 в главе 3 «Особенности работы с устройствами, подключенными к шине PCI») контроллера PCI IDE имеют следующее назначение:

- бит 0 — режим работы первичного канала (0 — режим совместимости, 1 — режим чистой PCI). Содержимое данного разряда имеет значение только в том случае, если значение бита 1 равно нулю;
- бит 1 — признак поддержки первичным каналом обоих режимов работы (0 — поддерживается только режим, определяемый битом 0, 1 — поддерживаются оба режима);
- бит 2 — режим работы вторичного канала (0 — режим совместимости, 1 — режим чистой PCI). Содержимое данного разряда имеет значение только в том случае, если значение бита 3 равно нулю;
- бит 3 — признак поддержки вторичным каналом обоих режимов работы (0 — поддерживается только режим, определяемый битом 2, 1 — поддерживаются оба режима);
- биты 4–6 — не используются (должны быть установлены в 0);
- бит 7 — признак поддержки режима Bus Master (0 — режим не поддерживается, 1 — режим поддерживается).

Для того чтобы можно было работать с дисками в режиме DMA, контроллер PCI должен иметь возможность захватывать шину, то есть должен поддерживать режим Bus Master IDE. Так как контроллеры современных материнских плат поддерживают по обоим каналам одинаковые режимы, значение байта интерфейса может быть равно 80h, 85h или 8Ah. Соответственно поиск контроллера нужно проводить по трем возможным кодам класса: 010180h, 010185h и 01018Ah.

С регистрами конфигурационного пространства PCI программист обычно не работает — программирование контроллера PCI IDE и моста PCI-to-ISA возлагается на операционную систему. На современных системных платах настройку временных параметров циклов PIO и DMA для установленных в системе дисков осуществляет при включении компьютера процедура начальной загрузки BIOS.

Однако, кроме регистров шины PCI, у контроллера имеется также набор обычных регистров ввода-вывода — через них осуществляется настройка контроллера PCI DMA на область оперативной памяти, с которой диск должен осуществлять обмен данными. Описание блока регистров ввода-вывода PCI IDE приведено в табл. 6.46 (все

регистры доступны как для записи, так и для считывания информации). Адрес блока регистров хранится в регистре базового адреса, размещенном в конфигурационном пространстве PCI IDE со смещением 20h (разряды 4–15 этого регистра хранят разряды 4–15 базового адреса блока, а разряды 0–3 базового адреса всегда равны 0; базовый адрес может принимать значения от 300h до FFF0h). Чтобы определить базовый адрес блока регистров, нужно прочитать значение (16-разрядное слово) из конфигурационного регистра со смещением 20h и выделить адрес командой AND со значением FFF0h.

Таблица 6.46. Блок регистров ввода-вывода PCI IDE

Смещение от базового адреса	Размер	Мнемоническое обозначение	Назначение
00h	BYTE	BMICP	Командный регистр первичного канала IDE
01h	BYTE	—	Зарезервирован
02h	BYTE	BMISP	Регистр состояния первичного канала IDE
03h	BYTE	—	Зарезервирован
04h	DWORD	BMIDTPP	Указатель на таблицу дескрипторов первичного канала IDE
0Bh	BYTE	BMICS	Командный регистр вторичного канала IDE
09h	BYTE	—	Зарезервирован
0Ah	BYTE	BMISS	Регистр состояния вторичного канала IDE
0Bh	BYTE	—	Зарезервирован
0Ch	DWORD	BMIDTPS	Указатель на таблицу дескрипторов вторичного канала IDE

Формат командного регистра канала показан на рис. 6.20. Разряды регистра имеют следующее назначение:

- бит 0 — управление работой Bus Master (пуск/останов: 1 — начать передачу в режиме DMA, 0 — остановить передачу);
- биты 1 и 2 — зарезервированы (должны быть установлены в 0);
- бит 3 — направление передачи данных (0 — чтение данных, 1 — запись данных);
- биты 4–7 — зарезервированы (должны быть установлены в 0).

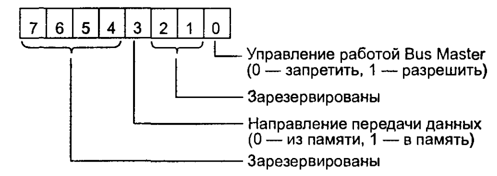


Рис. 6.20. Формат командного регистра канала PCI IDE

Формат регистра состояния канала приведен на рис. 6.21. Назначение разрядов регистра:

- бит 0 — признак активности Bus Master IDE (этот разряд принимает значение 1, если установлен бит 0 командного регистра);
- бит 1 — признак ошибки передачи данных (устанавливается в 1 при возникновении ошибки; сбрасывается программно путем записи в этот разряд единицы);
- бит 2 — признак прерывания (устанавливается в 1 при поступлении прерывания от устройства IDE; сбрасывается программно путем записи в этот разряд единицы);
- биты 3 и 4 — зарезервированы (должны быть установлены в 0);
- бит 5 — индикатор способности диска 0 работать в режиме DMA (данный разряд программно устанавливается в 1 драйвером диска или BIOS при выполнении самотестирования в процессе начальной загрузки системы, если Master-диск поддерживает режим DMA и параметры канала оптимизированы на максимальную скорость передачи);
- бит 6 — индикатор способности диска 1 работать в режиме DMA (данный разряд программно устанавливается в 1 драйвером диска или BIOS при выполнении самотестирования в процессе начальной загрузки системы, если Slave-диск поддерживает режим DMA и параметры канала оптимизированы на максимальную скорость передачи);
- бит 7 — признак симплексного режима (0 — первичный и вторичный каналы независимы друг от друга и могут работать одновременно; 1 — в каждый момент времени допускается работа только одного из каналов).

Регистр указателя на таблицу дескрипторов канала должен содержать линейный (абсолютный) 32-разрядный адрес таблицы. Значение адреса выравнивается на двойное слово, то есть два младших разряда регистра обязательно содержат нули.



Рис. 6.21. Формат регистра состояния канала PCI IDE

Чтобы контроллер мог осуществлять передачу данных, необходимо указать ему область оперативной памяти, с которой должна выполняться эта операция. Область памяти задается при помощи таблицы дескрипторов физических областей памяти PRDT (Physical Region Descriptor Table). Таблица состоит из одного или нескольких 8-байтных дескрипторов PRD (Physical Region Descriptor), описывающих используемые при передаче данных участки памяти. Адрес таблицы должен быть выровнен на двойное слово (младшие два бита адреса всегда нулевые); размер таблицы не должен превышать 64 Кбайт, то есть таблица может содержать до 8192 элементов.

Формат дескриптора PRD приведен в табл. 6.47. Физический адрес участка памяти выравнивается на слово, то есть должен быть четным 32-разрядным числом. Счетчик байт должен быть четным 16-разрядным числом. В слове признака конца таблицы используется только старший разряд (остальные разряды зарезервированы и должны быть установлены в 0), который устанавливается в 1 у последнего элемента таблицы, а у остальных элементов равен 0. Суммарный размер всех описанных в таблице PRDT областей не должен быть меньше объема данных, передача которых задана командой, посланной диску. Более детально формат дескриптора изображен на рис. 6.22.

Таблица 6.47. Формат дескриптора PRD (элемента таблицы PRDT)

Смещение	Размер	Назначение
00h	DWORD	Физический адрес участка памяти
04h	WORD	Размер участка памяти (счетчик байтов)
06h	WORD	Признак конца таблицы

Байт 7	Байт 6	Байт 5	Байт 4	Байт 3	Байт 2	Байт 2	Байт 1	Байт 0
63	48	47	32	31				0
Зарезервировано		Счетчик байтов		Физический адрес области памяти				

Признак конца таблицы — Младший разряд всегда равен 0

Таблица PRDT

Рис. 6.22. Формат дескриптора PRD

Физические адреса участков памяти могут быть заданы произвольным образом, то есть описываемые дескрипторами участки могут быть не смежными и размещаться в произвольном порядке (рис. 6.23).

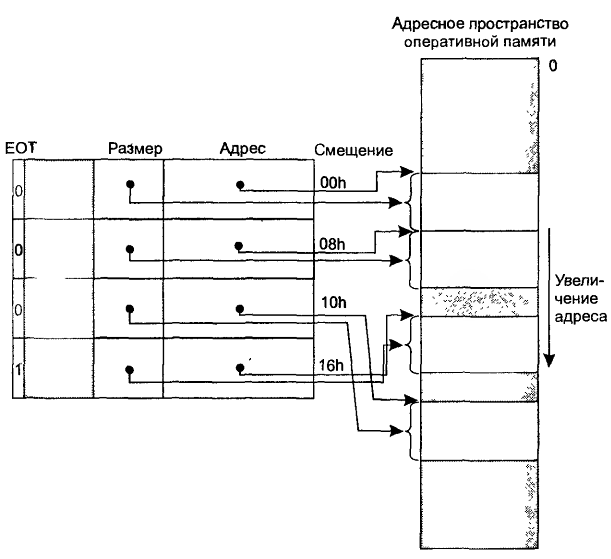


Рис. 6.23. Адресация памяти при помощи таблицы PRDT

В поле счетчика байт может находиться любое четное число от 0 до FFFEh, причем 0 соответствует значению 10000h, то есть можно задавать размер участка памяти от 2 байт до 64 Кбайт. На практике значение счетчика байтов выбирается равным размеру 16-разрядного сегмента (64 Кбайт), размеру страницы свопинга (4 Кбайт) или

размеру одной из дисковых структур (сектора или кластера жесткого диска). Размер всех используемых участков, как правило, одинаковый.

При выполнении операции передачи данных между памятью и диском в режиме DMA действия рекомендуется выполнять в указанном ниже порядке.

1. Подготовить таблицу PRD в оперативной памяти.
2. Загрузить адрес таблицы PRD в регистр указателя на таблицу дескрипторов соответствующего канала (канала, к которому подключен диск).
3. Установить значение бита направления передачи данных в командном регистре используемого канала в соответствии с выполняемой операцией.
4. Стереть бит прерывания и бит ошибки в регистре состояния используемого канала.
5. Послать диску команду, инициирующую требуемую операцию считывания или записи данных в режиме DMA.
6. Включить режим Bus Master путем установки в 1 бита управления режимом работы в командном регистре соответствующего канала.
7. Ожидать завершения передачи данных между диском и памятью (после завершения передачи диск выдает сигнал прерывания, что приводит к установке в 1 бита признака прерывания в регистре состояния).
8. Отключить режим Bus Master, сбросив в 0 бит управления режимом в командном регистре.
9. Проверить значения в регистрах состояния контроллера и диска, чтобы удостовериться в успешном завершении операции.

В листинге 6.12 приведен пример программы `IDE_DMA_Test`, осуществляющей считывание в режиме DMA загрузочного сектора Master-диска канала 1 (программа рассчитана на стандартный способ подключения жесткого диска к персональному компьютеру). Кроме процедур ввода-вывода общего назначения, описанных в данной главе и главе 1 «Работа с клавиатурой», программа использует также следующие подпрограммы:

- подпрограмма `SearchBusMasterIDEContr` осуществляет поиск контроллера PCI IDE по коду класса, а затем определяет адрес блока регистров ввода-вывода PCI DMA;

- подпрограмма HDD_Presence_Test проверяет, подключен ли диск 0 к каналу 1;
- подпрограмма ReadBootSector подготавливает канал к работе в режиме DMA, посылает диску команду считывания сектора в режиме DMA, а затем отображает сектор на экран в ASCII-коде;
- подпрограмма FatalError осуществляет вывод на экран сообщения об ошибке, ожидает нажатия любой клавиши, а затем выполняет аварийное завершение работы программы.

Листинг 6.12. Считывание загрузочного сектора Master-диска канала 1 в режиме DMA

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Текстовые сообщения
Txt0 DB LIGHTCYAN,0,17
     DB "ТЕСТИРОВАНИЕ РЕЖИМА DMA MASTER-ДИСКА КАНАЛА 1",0
     DB YELLOW,5,8,"Содержимое загрузочного сектора "
     DB "Master-диска канала 1:",0
AnyK DB YELLOW,24,29,"Нажмите любую клавишу",0
; Сообщения об отсутствии компонентов системы,
; необходимых для запуска примера
NoPCI  DB LIGHTRED,12,18
       DB "Система не поддерживает интерфейс PCI BIOS",0
NoIDEC DB LIGHTRED,12,26
       DB "Контроллер PCI IDE не найден",0
BadReg DB LIGHTRED,12,28,"Неверный номер регистра",0
NoHDD  DB 12,22,"Master-диск на канале 1 не обнаружен",0
NoLBA  DB 12,25,"Диск не поддерживает режим LBA",0
; Сообщения об ошибках при выполнении считывания
DErr1  DB 12,8,"Тайм-аут: превышена допустимая "
       DB "длительность выполнения операции",0
DErr2  DB 12,25,"Неверный код режима адресации",0
DErr3  DB 12,27,"Неправильный номер канала",0
DErr4  DB 12,27,"Неправильный номер диска",0
DErr5  DB 12,26,"Неправильный номер головки",0
DErr6  DB 12,17,"Ошибка при выполнении "
       DB "команды жестким диском.",0
```

```
; Адрес блока регистров контроллера PCI IDE
IDEContrRegsBaseAddr DW ?
ENDS

; Буфер для работы с диском в режиме DMA
SEGMENT DMA_BUF para public 'DATA'
        DB 8000h DUP(?)
ENDS

; Буфер для таблицы PRD
SEGMENT PRD_Table para public 'DATA'
        DB 1024 DUP(?)
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC IDE_DMA_Test
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести заголовок на экран
        MShowColorText 2,Txt0
; Задать номер канала и номер диска
        mov     [ChannelNumber],1
        mov     [HDDNumber],0
; Проверить наличие и работоспособность диска
        call    HDD_Presence_Test
; Найти контроллер PCI IDE
        call    SearchBusMasterIDEContr
; Создаем таблицу PRD, состоящую из одной записи
        mov     AX,PRD_Table
        mov     ES,AX
        ; Вычислить и записать адрес области данных
        xor     EAX,EAX
        mov     AX,DMA_BUF
        shl     EAX,4
        mov     [ES:0],EAX
```

продолжение

Листинг 6.12 *(продолжение)*

```
        ; Записать количество передаваемых байтов данных
        mov     [word ptr ES:4],512
        ; Установить признак последней строки таблицы
        mov     [word ptr ES:6],8000h
; Загрузить физический адрес таблицы PRD в
; соответствующий регистр контроллера PCI IDE
        ; Вычислить абсолютный адрес PRDT (умножить
        ; номер сегмента PRD_Table на 16)
        xor     EAX,EAX
        mov     AX,PRD_Table
        shl     EAX,4
        ; Вычислить адрес регистра контроллера
        mov     DX,[IDEContrRegsBaseAddr]
        add     DX,04h
        ; Записать адрес PRD в регистр
        out     DX,EAX
; Прочитать загрузочный сектор диска в режиме DMA
        call    ReadBootSector

; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP IDE_DMA_Test

;*****************************************************
;*   НАЙТИ КОНТРОЛЛЕР PCI IDE И ОТОБРАЗИТЬ НА ЭКРАН    *
;* СОДЕРЖИМОЕ РЕГИСТРОВ КОНФИГУРАЦИОННОГО ПРОСТРАНСТВА *
;*****************************************************
PROC SearchBusMasterIDEContr near
        pushad
; Проверить наличие PCI BIOS
        mov     AX,0B101h
        int     1Ah
        jc      @@PCIBIOSNotFound
        cmp     EDX,20494350h
        jne     @@PCIBIOSNotFound

; Найти контроллер Bus Master IDE по коду класса
        mov     AX,0B103h
        mov     ECX,010180h  ;первый вариант кода
        mov     SI,0
        int     1Ah
        jnc     @@ReadPCIRegisters ;устройство найдено
        mov     AX,0B103h
        mov     ECX,010185h  ;второй вариант кода
        mov     SI,0
```

```
        int     1Ah
        jnc     @@ReadPCIRegisters ;устройство найдено
        mov     AX,0B103h
        mov     ECX,01018Ah  ;третий вариант кода
        mov     SI,0
        int     1Ah
        ; Устройство найдено?
        jc      @@DeviceNotFound  ;устройство не найдено

; Устройство обнаружено, его координаты на шине PCI
; находятся в регистре BX
@@ReadPCIRegisters:
; Получить базовый адрес блока регистров контроллера IDE
        mov     AX,0B10Ah ;читать двойное слово
        mov     DI,20h    ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        and     CX,0FFF0h ;сбросить младшие 4 разряда
        mov     [IDEContrRegsBaseAddr],CX
        popad
        ret
; Обработка ошибок
@@BadRegisterNumber:
        MFatalError BadReg ;неверный номер регистра
@@DeviceNotFound:
        MFatalError NoIDEC ;не найден контроллер IDE
@@PCIBIOSNotFound:
        MFatalError NoPCI  ;не поддерживается PCI BIOS
ENDP SearchBusMasterIDEContr

;*****************************************************
;* ПРОВЕРИТЬ ПРИСУТСТВИЕ ДИСКА И НАЛИЧИЕ ПОДДЕРЖКИ LBA *
;*****************************************************
PROC HDD_Presence_Test near
        pusha
; Опросить Master-диск канала 1
        call    ReadHDD_ID
        ; Диск подключен?
        cmp     [HDDErrorCode],0
        jne     @@DiskNotFound
        ; Режим LBA поддерживается?
        cmp     [dword ptr SectorDataBuffer+60*2],0
        je      @@LBANotSupported
        popa
        ret
; Обработка ошибок
@@DiskNotFound:
        MFatalError NoHDD ;диск не найден
@@LBANotSupported:
        MFatalError NoLBA ;не поддерживается LBA
```

продолжение ⇨

Листинг 6.12 (*продолжение*)

```
ENDP HDD_Presence_Test

;******************************************
;* ПРОЧИТАТЬ ЗАГРУЗОЧНЫЙ СЕКТОР В РЕЖИМЕ DMA *
;******************************************
PROC ReadBootSector near
        pushad
; Сбросить разряды ошибки и прерывания в регистре
; состояния канала 1
        mov     DX,[IDEContrRegsBaseAddr]
        add     DX,02h
        mov     AL,110b
        out     DX,AL
; Настроить контроллер DMA на запись данных в память
; по каналу 1
        ; Загрузить адрес управляющего регистра канала 1
        mov     DX,[IDEContrRegsBaseAddr]
        ; Сбросить управляющий регистр
        mov     AL,0
        out     DX,AL
        ; Установить бит направления передачи данных
        mov     AL,1000b
        out     DX,AL

; Послать команду чтения сектора 0 (загрузочного)
        ; Подготовить параметры команды
        mov     [ATAAddressMode],1 ;режим LBA
        mov     [dword ptr SectorAddress],0    ;сектор 0
        mov     [dword ptr ATASectorNumber],0
        mov     [ATAFeatures],0
        mov     [ATASectorCount],1 ;прочесть один сектор
        mov     EAX,[SectorAddress]
        ; Послать команду диску
        mov     [ATACommand],0CBh  ;чтение в режиме DMA
        call    SendCommandToHDD
        ; Проверить код ошибки
        cmp     [HDDErrorCode],0   ;имелась ошибка?
        jne     @@Error            ;обработка ошибки

; Активизировать канал 1 контроллера DMA
        mov     DX,[IDEContrRegsBaseAddr]
        mov     AL,1001b
        out     DX,AL

; Ожидать готовности данных HDD
        mov     AX,0
        mov     ES,AX
        mov     DX,[HDDBasePortAddr]
        add     DX,7     ;адрес регистра состояния
```

```
@@WaitCompleet:
        ; Проверить время выполнения команды
        mov     EAX,[ES:046Ch]
        sub     EAX,[HDDTime]
        cmp     EAX,MaxHDDWaitTime
        ja      @@Err1   ;ошибка тайм-аута
        ; Проверить готовность
        in      AL,DX
        test    AL,88h   ;состояние сигнала BSY и DRQ
        jnz     @@WaitCompleet
; Ожидать конца цикла DMA
        mov     DX,[IDEContrRegsBaseAddr]
        add     DX,02h
@@WaitDMAOperationEnd:
        in      AL,DX
        test    AL,100b
        jnz     @@WaitDMAOperationEnd

; Сбросить управляющий регистр канала 1 контроллера DMA
        mov     DX,[IDEContrRegsBaseAddr]
        mov     AL,0
        out     DX,AL

; Отобразить на экран содержимое прочитанного сектора
; в ASCII-кодах
        push    DS
        mov     AX,DMA_BUF
        mov     DS,AX
        mov     AX,0B800h
        mov     ES,AX
        mov     DI,7*160+8*2
        xor     SI,SI
        ; Задать светло-голубой цвет и синий фон
        mov     AH,LIGHTCYAN+BLUE*16
; Цикл по строкам
        mov     DX,8  ;счетчик строк
@@OutNextString:
; Цикл по символам строки
        mov     CX,64 ;счетчик символов в строке
@@OutNextChar:
        lodsb
        stosw
        loop    @@OutNextChar
        add     DI,16*2
        dec     DX
        jnz     @@OutNextString
        pop     DS
; Ожидать нажатия любой клавиши и выйти из процедуры
        MShowColorString AnyK
        call    GetChar
        popad
```

продолжение ⇨

Листинг 6.12 (*продолжение*)

```
        ret
; Обработка ошибок
@@Error:
        cmp     [HDDErrorCode],1
        je      @@Err1
        cmp     [HDDErrorCode],2
        je      @@Err2
        cmp     [HDDErrorCode],3
        je      @@Err3
        cmp     [HDDErrorCode],4
        je      @@Err4
        cmp     [HDDErrorCode],5
        je      @@Err5
        cmp     [HDDErrorCode],6
        je      @@Err6
@@Err1: MFatalError DErr1
@@Err2: MFatalError DErr2
@@Err3: MFatalError DErr3
@@Err4: MFatalError DErr4
@@Err5: MFatalError DErr5
@@Err6: MFatalError DErr6
ENDP ReadBootSector
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуру непосредственного считывания
; сектора с жесткого диска
include "list6_07.inc"

END
```

ПРИМЕЧАНИЕ

Для успешного запуска программы необходимо, чтобы жесткий диск, способный работать в режиме UDMA, был подключен к каналу 1 в качестве Master-диска. Кроме того, контроллер PCI сиотемной платы должен поддерживать режим Bus Master, а система BIOS должна обеспечивать начальную установку временных параметров цикла DMA в процессе начальной загрузки системы.

Риск потери информации, связанный с выполнением операций форматирования и записи данных

Непосредственная работа с контроллером дисков чаще всего применяется для записи или считывания данных в режиме реального

времени — например, при проведении экспериментов или при обработке видеоизображений. Режим реального времени предполагает непрерывную передачу информации — система должна успевать обрабатывать каждый пакет данных до поступления следующего пакета (иначе один из пакетов будет потерян). При работе с дисковыми накопителями непрерывность передачи информации может нарушаться при периодической термокалибровке привода головок, при выполнении процедуры самотестирования либо в результате фрагментации файлов. Для работы в реальном времени, вообще говоря, выпускаются специальные жесткие диски (например, мультимедийные цифровые видеомагнитофоны), но можно попробовать использовать и обычные диски с большим объемом буферной памяти, которая позволяет замаскировать термокалибровку и самотестирование. Альтернативным вариантом является создание буфера данных большого объема в оперативной памяти компьютера.

Поскольку непосредственная работа с контроллером резко увеличивает риск разрушения данных на диске при отладке программы в результате ошибок программиста, желательно максимально изолировать от остальных данных тот участок, с которым идет работа напрямую через регистры контроллера. Идеальным, но иногда недостижимым (по финансовым причинам) вариантом является выделение для работы в реальном времени отдельного жесткого диска; если невозможно выделить целый диск, то выделяется раздел диска, обычно — в конце адресного пространства (чтобы уменьшить риск стирания последующих разделов при ошибке адресации). В обоих случаях вместо стандартной разметки диска по методу FAT можно применить свою собственную, упрощенную организацию данных.

Непосредственная работа на уровне контроллера с обычными файлами произвольного типа и размера, произвольным образом размещенными на диске, требует исключительной аккуратности. Программист в этом случае должен очень хорошо знать все особенности файловой системы, которую он использует; прежде чем начинать отладку процедур для записи данных на диск, он должен хорошо освоить адресацию данных в режиме считывания. Ошибки адресации опасны тем, что могут постепенно разрушать информацию на диске, оставаясь при этом незамеченными в течение длительного времени. Если данные непременно должны быть записаны в виде файла в стандартный DOS-раздел, то, опять-таки, можно выделить для него неперемещаемую область (для чего файл нужно пометить как системный), желательно — фиксированного размера и в конце раздела: файл лучше при этом записывать в корневой каталог.

Выполнение операций низкоуровневого форматирования связано с очень серьезным риском. Не применяйте команды форматирования жестких дисков ни из набора прерываний DOS и BIOS, ни из набора ATA. Ни в коем случае не запускайте процедуру низкоуровневого форматирования из BIOS SETUP. Каждая фирма-изготовитель дисков выпускает для них свои собственные утилиты — следовательно, фактически отсутствует единый стандарт выполнения операций форматирования.

В нашей стране компьютеры обычно эксплуатируются до полного физического износа, то есть до тех пор, пока ремонт не становится невозможным. Мне приходилось так или иначе иметь дело примерно с полутора сотнями персональных компьютеров, поэтому для сбора статистики отказов имеется не слишком много данных, но определенные закономерности можно выявить. Жесткий диск заканчивает свой жизненный путь двумя способами — либо сгорает аппаратура (мотор или встроенный контроллер), либо изнашивается поверхность диска и появляется много дефектных секторов. В последнем случае диск окончательно погибает, когда кто-нибудь запускает процедуру низкоуровневого форматирования из BIOS SETUP или универсальную форматирующую программу. Отформатированный такими способами диск часто не поддается восстановлению даже при помощи дисковых утилит фирмы-изготовителя. Жесткие диски в последнее время сильно подешевели, и если диск сильно изношен, не тратьте время на восстановление, а просто замените его. Кроме того, информация, содержащаяся на диске большого объема, стоит обычно гораздо больше самого диска — хранить данные на ненадежных дисках экономически неэффективно (если не верите, то попробуйте подсчитать трудозатраты на одну только запись информации на диск).

Все основные поставщики жестких дисков предоставляют бесплатно через Интернет программы для обслуживания своих изделий, в том числе — для низкоуровневого форматирования. Обычно эти программы размещаются на основном сервере или сервере технической поддержки:

- Fujitsu — http://www.fujitsu.com или http://www.fujitsu.com.tw;
- IBM — http://www.storage.ibm.com;
- Maxtor — http://www.maxtor.com;
- Quantum — http://www.quantum.com;
- Samsung — http://www.samsung.com;
- Seagate — http://www.seagate.com;
- Western Digital — http://www.wdc.com.

Глава 7 Принтеры: печать в растровом режиме

У программистов при работе с принтерами возникает та же самая проблема, что и при работе с другими типами аппаратуры: изготовители в документации обычно приводят наборы команд, но не дают ни пояснений, зачем эти команды нужны, ни рекомендаций по использованию, ни примеров — предполагается, что все и так понятно из названия команды. Есть только одна особенность: другие устройства еще не документированы, а принтеры — уже не документированы. В последнее время изготовители принтеров стали придерживаться теории, утверждающей, что пользователям знать о командах вообще ничего не нужно, и изъяли всякое упоминание о них из описаний типа User's Manual, поставляемых вместе с принтерами. В общем-то, эта теория верна, однако для специалистов информацию можно было оставить хотя бы в Интернете...

В литературе информация по работе с принтерами представлена в форме отдельных фрагментов. Книги, рассчитанные на инженеров-полиграфистов, много внимания уделяют работе со шрифтами, поскольку до сих пор приходится сталкиваться с проблемой адаптации печатающих устройств к национальным языковым особенностям, то есть к алфавитам разных стран мира. Для архитекторов и проектировщиков необходима векторная графика — она тоже описана неплохо.

Растровая печать — это простой и полезный графический режим, он применяется во многих программных пакетах для вывода на принтер рисунков и фотографий, однако внимания данному режиму в литературе уделялось мало по целому ряду причин. Во-первых, режим простой, и изготовители принтеров считают, что все и так ясно. Во-вторых, потребность в создании собственных программ, непосредственно работающих с растром, возникает только у небольшой

группы потребителей — у инженеров и ученых, занимающихся измерением и регистрацией параметров процессов, протекающих в реальном времени. В-третьих, до недавнего времени у персональных компьютеров было недостаточно оперативной памяти для эффективной работы с растром: создание черно-белого растрового изображения размером с один лист бумаги формата А4 при разрешении 300 точек на дюйм (12 тчк/мм) требует 1,1 Мбайт памяти. При полутоновой или цветной печати для кодирования каждой точки требуется не один, а несколько битов памяти, то есть для создания изображения размера А4 нужно уже от 5 до 30 Мбайт.

Хотя идея растрового режима, то есть разделения изображения на клеточки, сама по себе очень проста, реализовать эту идею при недостаточном объеме памяти компьютера довольно трудно: приходится разбивать лист на небольшие кусочки — сегменты — и работать с ними, по очереди считывая и записывая обратно на диск. Поскольку элементы изображения не обязательно находятся внутри одного сегмента, приходится прорисовывать их по частям, контролируя пересечение границ сегментов. Сегментация изображения, таким образом, сильно замедляет и затрудняет работу с ним. Значительное падение цен на оперативную память и увеличение ее типового объема в персональных компьютерах до 32 Мбайт открывает новые возможности для использования растрового режима — теперь изображение можно создавать сразу для целого листа, что существенно упрощает алгоритмы рисования.

Основной проблемой растрового режима является отсутствие единого общепризнанного командного языка: для лазерных принтеров фактическим стандартом является язык команд PCL фирмы Hewlett-Packard, а для матричных — язык ESC/P2 фирмы Epson. Для струйных принтеров единого стандарта нет: принтеры Hewlett-Packard (не все) работают с языком PCL, принтеры Epson — с языками ESC/P2 и Epson raster, а остальные имеют собственные командные языки, разработанные изготовителями и не описанные в общедоступной документации.

И у Epson, и у Hewlett-Packard очень скудно представлены сведения о цветной печати: в лучшем случае приводится краткое описание набора команд, а примеры их использования отсутствуют. Ситуация с документацией тем более странная, что для современных принтеров нижней ценовой группы растровый режим является основным, а во многих случаях и единственным режимом работы.

Вывод информации на принтер при помощи стандартных функций BIOS

Для работы с принтером предназначена группа функций BIOS, вызываемых по прерыванию Int 17h. После выполнения любой из функций данной группы в регистре AH будет возвращен код состояния принтера, разряды которого имеют следующее значение:

- бит 0 — признак тайм-аута (0 — нормальное состояние, 1 — ошибка тайм-аута, то есть принтер не отвечает);
- биты 1 и 2 — не используются, установлены в 0;
- бит 3 — признак ошибки ввода-вывода (0 — ошибка, 1 — нет ошибки);
- бит 4 — признак выбора принтера (0 — принтер в автономном режиме, 1 — принтер в режиме подключения);
- бит 5 — контроль наличия бумаги (0 — бумага вставлена, 1 — нет бумаги);
- бит 6 — подтверждение приема (0 — подтверждение приема символа, 1 — обычное состояние);
- бит 7 — признак занятости принтера (0 — принтер занят, 1 — принтер свободен).

Формат байта кода состояния принтера показан на рис. 7.1.

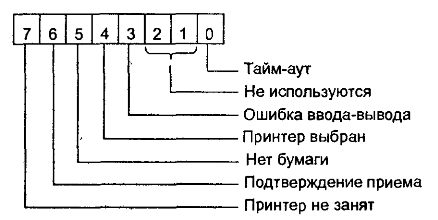


Рис. 7.1. Формат байта кода состояния

Прерывание Int 17h, функция 00h: вывести символ на принтер

Функция предназначена для выполнения побайтного вывода информации. Это основная функция данной группы — она обеспечивает посылку команд и данных на принтер.

Перед вызовом прерывания требуется записать в регистры следующую информацию:

- в AH — значение 00h;
- в AL — код выводимого символа;
- в DX — номер порта принтера (0 – LPT1, 1 — LPT2, 2 – LPT3).

Прерывание Int 17h, функция 01h: инициализировать порт

Функция предназначена для выполнения инициализации (сброса) интерфейса принтера. Вызывают данную функцию после обнаружения серьезных сбоев в работе принтера.

Перед вызовом прерывания требуется записать в регистры следующую информацию:

- в AH — значение 01h;
- в DX — номер порта принтера (0 – LPT1, 1 – LPT2, 2 – LPT3).

Прерывание Int 17h, функция 02h: получить состояние принтера

Функция возвращает текущее состояние принтера. Обычно ее вызывают перед началом печати очередной строки или нового листа, чтобы проверить готовность принтера к печати: включен ли принтер и заправлена ли в него бумага.

Перед вызовом прерывания требуется записать в регистры следующую информацию:

- в AH — значение 02h;
- в DX — номер порта принтера (0 – LPT1, 1 – LPT2, 2 – LPT3).

Использование стандартных функций прерывания Int 17h

Независимо от того, в каком режиме осуществляется печать и какой командный язык при этом используется, процесс вывода информации заключается в побайтной передаче команд и данных с компьютера на принтер. Перечисленные выше функции используют режим передачи стандартного параллельного порта (режим SPP), который является самым медленным, но зато поддерживается всеми моделями принтеров.

В листинге 7.1 приведены процедуры, предназначенные для работы с принтером, подключенным к первому параллельному порту (LPT1):

- процедура OutCharToLPT1 осуществляет вывод символа в порт LPT1 при помощи функций BIOS;
- процедура OutCommandToLPT1 использует подпрограмму OutCharToLPT1 для подачи на принтер командной последовательности символов (Esc-последовательности).

Реакция на ошибки, возникающие при обмене данными с принтером, в указанных функциях реализована примитивно: при любой ошибке выдается сообщение о том, что принтер не готов к печати, а затем происходит аварийное завершение работы программы (выход в DOS).

Листинг 7.1. Процедуры для вывода символа и посылки команды на принтер LPT1

```
DATASEG
PErrTxt DB 12,27,"Принтер не готов к печати",0
ENDS

CODESEG
;****************************
;* ВЫВЕСТИ СИМВОЛ НА ПРИНТЕР *
;* Параметры:                *
;* AL - код символа.         *
;****************************
PROC OutCharToLPT1 near
        pusha
; Вывести символ на печать
        mov     AH,0
        mov     DX,0
        int     17h
        test    AH,00101001b
        jnz     @@PrintingError
        popa
        ret
; ВЫДАТЬ СООБЩЕНИЕ ОБ ОШИБКЕ И ВЫЙТИ ИЗ ПРОГРАММЫ
@@PrintingError:
        ; Вывести сообщение об ошибке
        MFatalError PErrTxt
ENDP OutCharToLPT1

;****************************************
;*       ПОСЛАТЬ КОМАНДУ НА ПРИНТЕР      *
;* Параметры:                            *
;* DS:SI - указатель на строку команды.  *
;* Первый байт строки содержит количество *
```

продолжение ⇒

Листинг 7.1 (*продолжение*)

```
;* байтов команды, посылаемых на принтер. *
;*****************************************
PROC OutCommandToLPT1 near
        pusha
        cld
; Загрузить счетчик байтов команды в CX
        lodsb
        xor     CX,CX
        mov     CL,AL
@@OutNextByte:
        lodsb
        call    OutCharToLPT1
        loop    @@OutNextByte
        popa
        ret
ENDP OutCommandToLPT1
ENDS
```

Процедуры из листинга 7.1 будут использоваться ниже, в примерах, демонстрирующих печать в растровом режиме на различных типах принтеров.

СОВЕТ

В случае если принтер на вашем компьютере подключен к порту LPT2, нужно изменить номер порта в процедуре OutCharToLPT1, то есть перед вызовом прерывания поместить в регистр DX значение 1 вместо 0.

Функции EPP BIOS

Когда получили массовое распространение периферийные устройства, выполняющие обработку документов (печать или сканирование) в растровом режиме, возникла потребность в значительном увеличении скорости передачи данных через параллельный порт. Две группы разработчиков практически одновременно выдвинули два различных стандарта [51, 61]:

- улучшенный параллельный порт (Enhanced Parallel Port, сокращенно EPP);
- порт с расширенными возможностями (Extended Capability Port, сокращенно ECP).

Оба стандарта имеют свои преимущества, поэтому разработчикам оборудования пришлось в одном контроллере параллельного порта реализовать поддержку сразу трех режимов — SPP, EPP и ECP.

Режим EPP применяется для работы со сканерами и внешними дисководами, а режим ECP — для работы с лазерными и струйными принтерами.

Стандарт EPP предусматривает возможность соединения подключаемых к параллельному порту устройств в цепочки, для чего EPP-устройство снабжается двумя 25-контактными разъемами (входным и выходным) и специальным коммутатором, делающим устройство «невидимым» для компьютера, когда оно не используется. В одну цепочку можно включить до 8 EPP-устройств; кроме того, в конце цепочки может присутствовать устройство, не поддерживающее стандарт EPP (например, принтер, работающий только в режимах SPP и ECP).

Цепочку устройств можно подключить к компьютеру напрямую или через мультиплексор. Мультиплексор может иметь до восьми выходных разъемов, так что общее количество подключенных устройств может достигать 72 (восемь цепочек по девять устройств).

Рассмотрим дополнительный набор функций BIOS, предназначенный для обслуживания новых режимов работы и получивший наименование EPP BIOS [44, 85].

Прерывание Int 17h, функция 02h: проверить наличие EPP BIOS

Функция проверяет наличие EPP BIOS. В случае если EPP BIOS поддерживается системой, функция возвращает вектор («точку входа») для вызова функций EPP BIOS.

Перед вызовом прерывания требуется записать в регистры следующую информацию:

- в AH — значение 02h;
- в DX — номер параллельного порта (0 — LPT1, 1 — LPT2, 2 — LPT3);
- в AL — значение 0;
- в CH — значение 45h (символ E);
- в BL — значение 50h (символ P);
- в BH — значение 50h (символ P).

В случае успешного завершения операции функция возвращает:

- в AH — значение 0;
- в AL — значение 45h;

- в CX — значение 5050h;
- в паре регистров DX:BX — вектор EPP (точку входа EPP BIOS).

Для вызова всех остальных функций EPP BIOS применяется вектор точки входа EPP BIOS, возвращаемый данной функцией.

ПРИМЕЧАНИЕ

Кроме регистров, используемых для передачи параметров при вызове функций EPP и для возврата результатов, в функциях используется также регистр BX, содержимое которого не сохраняется (теряется после вызова функции).

Переход по вектору EPP, функция 00h: определить конфигурацию и возможности порта

Функция Query Config позволяет определить текущую конфигурацию параллельного порта с заданным номером.

Перед вызовом данной функции по вектору EPP требуется записать в регистры следующую информацию:

- в AH — значение 00h;
- в DL — номер параллельного порта (0 – LPT1, 1 – LPT2, 2 – PT3).

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в AL — уровень прерывания от EPP порта (может принимать значения в диапазоне от 0 до 15; код FFh означает, что прерывания портом не поддерживаются);
- в BH — номер версии EPP BIOS;
- в BL — возможности параллельного порта (бит 0 — признак наличия мультиплексора, бит 1 — признак наличия поддержки двунаправленного режима PS/2, бит 2 — признак наличия поддержки режима EPP 1.9, бит 3 – признак наличия поддержки режима ECP, бит 4 — зарезервирован, бит 5 — признак наличия поддержки Centronics FIFO, бит 6 — признак наличия поддержки режима EPP 1.7, бит 7 — зарезервирован);
- в CX — базовый адрес группы регистров порта при работе в режиме SPP;
- в паре регистров ES:DI — указатель на ограниченную нулем текстовую строку, содержащую информацию о разработчике данной версии EPP BIOS.

Переход по вектору EPP, функция 01h: установить режим работы порта

Функция Set Mode позволяет установить режим работы параллельного порта с заданным номером. Вызов данной функции разрешен только в том случае, если процессор работает в «реальном» режиме.

Перед вызовом данной функции требуется записать в регистры следующую информацию:

- в AH — значение 01h;
- в DL — номер параллельного порта;
- в AL — код устанавливаемого режима (бит 0 — установить «режим совместимости» SPP, бит 1 — установить двунаправленный режим SPP, бит 2 — установить режим EPP, бит 3 — установить режим ECP, бит 4 — зарезервирован, бит 5 — установить режим Centronics FIFO, бит 6 — установить режим EPP 1.7, бит 7 — зарезервирован).

После выполнения функция возвращает в регистре AH код ошибки.

Переход по вектору EPP, функция 02h: определить режим работы порта

Функция Get Mode позволяет определить текущий режим работы параллельного порта с заданным номером. Вызов данной функции разрешен только в том случае, если процессор работает в «реальном» режиме.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 02h;
- в DL — номер параллельного порта.

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в AL — код режима работы порта (бит 0 — признак режима совместимости, бит 1 — признак двунаправленного режима, бит 2 — признак режима EPP, бит 3 — признак режима ECP, бит 4 — зарезервирован, бит 5 — признак режима Centronics FIFO, бит 6 — признак режима EPP 1.7, бит 7 — признак того, что разрешена обработка прерываний в режиме EPP).

Переход по вектору EPP, функция 03h: управление прерываниями

Функция Interrupt Control позволяет включать и отключать обработку прерывания, связанного с заданным параллельным портом. Вызов данной функции разрешен только в том случае, если процессор работает в «реальном» режиме.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 03h;
- в DL — номер параллельного порта;
- в AH — управляющий код (0 — запретить прерывания от порта EPP, 1 — разрешить прерывания).

После выполнения функция возвращает в регистре AH код ошибки.

Переход по вектору EPP, функция 04h: инициализация

Функция EPP Reset позволяет осуществить инициализацию («сброс») устройства, подключенного к заданному параллельному порту.

Перед вызовом данной функции требуется записать в регистры следующую информацию:

- в AH — значение 04h;
- в DL — номер параллельного порта.

После выполнения функция возвращает в регистре AH код ошибки.

Переход по вектору EPP, функция 05h: запись адреса

Функция Address Write выполняет цикл записи адреса устройства.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 05h;
- в DL — номер параллельного порта;
- в AL — адрес устройства.

После выполнения функция возвращает в регистре AH код ошибки.

Переход по вектору EPP, функция 06h: считывание адреса

Функция Address Read выполняет цикл считывания адреса активного устройства.

Перед вызовом данной функции требуется записать в регистры следующую информацию:

- в AH — значение 06h;
- в DL — номер параллельного порта.

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в AL — адрес устройства и дополнительные данные.

Переход по вектору EPP, функция 07h: запись байта

Функция Write Byte выполняет вывод одного байта данных через порт данных EPP.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 07h;
- в DL — номер параллельного порта;
- в AL — байт данных.

После выполнения функция возвращает в регистре AH код ошибки.

Переход по вектору EPP, функция 08h: запись блока данных

Функция Write Block выполняет вывод блока данных из заданного буфера через порт данных EPP.

Перед вызовом функции требуется поместить в регистры следующую информацию:

- в AH — значение 08h;
- в DL — номер параллельного порта;
- в CX — размер передаваемого блока в байтах (значение 0 в данном регистре соответствует размеру блока, равному 64 Кбайт);

- в пару регистров ES:SI – указатель на область памяти (буфер), содержащую передаваемый блок данных.

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в CX — значение 0 в случае успешного выполнения передачи блока или количество не переданных байт в случае возникновения ошибки (сбоя) в процессе передачи.

ПРИМЕЧАНИЕ

Задавать значение 0 в регистре CX при вызове данной функции не рекомендуется: могут возникать проблемы совместимости, связанные с различными реализациями EPP BIOS.

Переход по вектору EPP, функция 09h: считывание байта данных

Функция Read Byte выполняет считывание одного байта данных из порта данных EPP.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 09h;
- в DL — номер параллельного порта.

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в AL — принятый байт данных.

Переход по вектору EPP, функция 0Ah: считывание блока данных

Функция Read Block выполняет считывание блока данных в заданный буфер через порт данных EPP.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 0Ah;
- в DL — номер параллельного порта;

- в CX — размер принимаемого блока в байтах (значение 0 в данном регистре соответствует размеру блока, равному 64 Кбайт);
- в пару регистров ES:DI — указатель на область памяти, предназначенную для размещения принимаемого блока данных.

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в CX — значение 0 в случае успешного завершения операции или количество не переданных байт в случае возникновения ошибки.

ПРИМЕЧАНИЕ

Задавать значение 0 в регистре CX при вызове данной функции не рекомендуется: могут возникать проблемы совместимости, связанные с различными реализациями EPP BIOS.

Переход по вектору EPP, функция 0Bh: запись адреса и считывание байта

Функция Address/Byte Read выполняет комбинированную операцию: устанавливает адрес устройства, а затем принимает от него байт данных.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 0Bh;
- в DL — номер параллельного порта;
- в AL — адрес устройства.

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в AL — принятый байт данных.

Переход по вектору EPP, функция 0Ch: запись адреса и байта данных

Функция Address/Byte Write выполняет комбинированную операцию: устанавливает адрес устройства, а затем передает этому устройству байт данных.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 0Ch;
- в DL — номер параллельного порта;
- в AL — адрес устройства;
- в DH — передаваемый байт данных.

После выполнения функция возвращает в регистре AH код ошибки.

Переход по вектору EPP, функция 0Dh: запись адреса и считывание блока данных

Функция Address/Block Read выполняет комбинированную операцию: устанавливает адрес устройства, а затем принимает от него блок данных.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 0Dh;
- в DL — номер параллельного порта;
- в AL — адрес устройства;
- в CX — размер принимаемого блока в байтах (значение 0 в данном регистре соответствует размеру блока, равному 64 Кбайт);
- в паре регистров ES:DI — указатель на область памяти, предназначенную для размещения принимаемого блока данных.

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в CX — значение 0 в случае успешного завершения операции; количество не переданных байт в случае возникновения ошибки.

ПРИМЕЧАНИЕ

Задавать значение 0 в регистре CX при вызове данной функции не рекомендуется: могут возникать проблемы совместимости, связанные с различными реализациями EPP BIOS.

Переход по вектору EPP, функция 0Eh: запись адреса и блока данных

Функция Address/Block Write выполняет комбинированную операцию: устанавливает адрес устройства, а затем передает ему блок данных.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 0Eh;
- в DL — номер параллельного порта;
- в AL — адрес устройства;
- в CX — размер передаваемого блока в байтах (значение 0 в данном регистре соответствует размеру блока, равному 64 Кбайт);
- в пару регистров ES:SI — указатель на область памяти, содержащую передаваемый блок данных.

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в CX — значение 0 в случае успешного завершения операции или количество не переданных байт в случае возникновения ошибки.

ПРИМЕЧАНИЕ

Задавать значение 0 в регистре CX при вызове данной функции не рекомендуется: могут возникать проблемы совместимости, связанные с различными реализациями EPP BIOS.

Переход по вектору EPP, функция 0Fh: захватить порт

Функция Lock Port позволяет упорядочить ввод и вывод данных через EPP-порт. Данная функция применяется для выбора порта конкретного устройства, если это устройство входит в состав цепочке устройств или подключено через мультиплексор.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 0Fh;
- в DL — номер параллельного порта;
- в BL — адрес устройства (биты 0–3 задают номер порта мультиплексора, биты 4–7 — номер устройства в цепочке).

После выполнения функция возвращает в регистре AH код ошибки.

Номер порта мультиплексора может принимать значения в диапазоне от 1 до 8, номер устройства в цепочке — также от 1 до 8.

При использовании в системе драйвера для работы с мультиплексором или цепочкой устройств вызов функции Lock Port завершается успешно только в том случае, если порт еще не захвачен.

Захват порта должен быть выполнен перед началом работы с устройством. Пока порт не захвачен, допускается выполнение следующих операций:

- Device Interrupt;
- Installation Check;
- Real time Mode;
- Rescan Daisy Chain;
- Set Product ID;
- Query Config;
- Query Daisy Chain;
- Query Device Port;
- Query Mux.

Переход по вектору EPP, функция 10h: освободить порт

Функция Unlock Port освобождает EPP-порт и позволяет его использовать драйверам других устройств.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 10h;
- в DL — номер параллельного порта;
- в BL — адрес устройства (биты 0–3 задают номер порта мультиплексора, биты 4–7 — номер устройства в цепочке).

После выполнения функция возвращает в регистре AH код ошибки.

Переход по вектору EPP, функция 11h: установить обработчик прерываний

Функция Device Interrupt позволяет драйверу EPP-устройства установить собственный обработчик прерывания для параллельного порта с заданным номером. Вызов данной функции разрешен только в том случае, если процессор работает в «реальном» режиме.

Перед вызовом функции требуется поместить в регистры следующую информацию:

- в AH — значение 11h;
- в DL — номер параллельного порта;
- в AL — код выполняемой операции (0 — запретить обработку прерываний, 1 — разрешить обработку прерываний, 2 — удалить обработчик прерываний);
- в пару регистров ES:DI — дальний указатель на обработчик прерываний;

- в BL — адрес устройства (биты 0–3 задают номер порта мультиплексора, биты 4–7 — номер устройства в цепочке).

После выполнения функция возвращает в регистре AH код ошибки.

Перед вызовом обработчика прерывания запрещаются. Обработчик не должен выполнять захват устройства, так как эта операция уже осуществлена драйвером. При выходе из обработчика нужно отправить инструкцию EOI контроллеру прерываний. Выход должен осуществляться при помощи инструкции IRET.

Переход по вектору EPP, функция 12h: режим реального времени

Функция Real Time Mode настраивает драйвер для работы в режиме реального времени. С помощью данной функции можно определить наличие устройств, требующих использования режима реального времени: если таких устройств нет, драйвер может передавать данные большими блоками; в противном случае должны использоваться маленькие блоки.

Перед вызовом функции требуется поместить в регистры следующую информацию:

- в AH — значение 12h;
- в AL — код выполняемой операции (0 — проверить наличие устройств, работающих в режиме реального времени; 1 — проинформировать драйвер о наличии устройства реального времени; 2 — сбросить флаг реального времени).

После выполнения функция возвращает в регистре AH код ошибки. При выполнении поиска устройств, работающих в режиме реального времени (код операции 0) в регистре AL будет возращен результат поиска (0 — устройства реального времени не обнаружены, 1 — найдено одно или несколько устройств).

Переход по вектору EPP, функция 40h: опросить мультиплексор

Функция Query Mux позволяет получить информацию о мультиплексоре, подключенном к указанному порту.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 40h;
- в DL — номер параллельного порта.

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в AL — флаги состояния (бит 0 — захват канала, бит 1 — наличие обработчика прерываний);
- в BL — номер выбранного (активного) порта мультиплексора;
- в BH — номер версии драйвера мультиплексора;
- в паре регистров ES:DI — указатель на ASCIIZ-строку, идентифицирующую разработчика драйвера.

Переход по вектору EPP, функция 41h: опросить устройство

Функция Query Device Port позволяет получить информацию об устройстве с заданным адресом, подключенном к указанному порту.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 41h;
- в DL — номер параллельного порта;
- в BL — адрес устройства (биты 0–3 задают номер порта мультиплексора, биты 4–7 — номер устройства в цепочке).

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в AL — флаги состояния (бит 0 — выбор порта, бит 1 — захват порта, бит 2 — прерывания разрешены, бит 3 — наличие обработчика прерываний);
- в CX — идентификатор устройства (ноль, если устройство не определено).

Переход по вектору EPP, функция 42h: задать идентификатор устройства

Функция Set Product ID позволяет задать идентификатор для устройства, которое не поддерживает циклы чтения адреса или не способно выдать собственный идентификатор.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 42h;
- в DL — номер параллельного порта;
- в BL — адрес устройства (биты 0–3 задают номер порта мультиплексора, биты 4–7 — номер устройства в цепочке);
- в CX — идентификатор для устройства.

После выполнения функция возвращает в регистре AH код ошибки.

Переход по вектору EPP, функция 50h: повторное сканирование цепочки устройств

Функция Rescan Daisy Chain Product используется для динамического перераспределения номеров портов между устройствами, соединенными в цепочку.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 50h;
- в DL — номер параллельного порта;
- в BL — номер порта мультиплексора (допускается использование номеров от 1 до 8; ноль означает отсутствие мультиплексора).

После выполнения функция возвращает в регистре AH код ошибки.

Переход по вектору EPP, функция 51h: задать идентификатор устройства

Функция Query Daisy Chain позволяет получить информацию о цепочке устройств, подключенной к указанному порту.

Перед вызовом функции требуется записать в регистры следующую информацию:

- в AH — значение 51h;
- в DL — номер параллельного порта.

После выполнения функции в регистрах находится следующая информация:

- в AH — код ошибки;
- в AL — флаги состояния (бит 0 — признак захвата канала, 1 — признак наличия обработчика прерываний);

- в BL — номер выбранного устройства;
- в CL — количество устройств в цепочке (0 — нет цепочки);
- в паре регистров ES:DI — указатель на ASCIIZ-строку, идентифицирующую разработчика драйвера.

Коды ошибок EPP BIOS

Значение кода ошибки, возвращаемого функциями EPP BIOS в регистре AH, расшифровывается следующим образом:

- 00h — успешное завершение операции, ошибок нет;
- 01h — тайм-аут;
- 02h — команда или операция не поддерживаются EPP BIOS;
- 03h — некорректный адрес порта устройства;
- 04h — EPP BIOS занята (BIOS не является реентерабельным);
- 05h — некорректный параметр;
- 10h — мультиплексор уже заблокирован (захвачен);
- 20h — мультиплексор отсутствует;
- 40h — программа обслуживания (менеджер) цепочки или мультиплексора не установлена;
- 41h — порт устройства заблокирован (захвачен);
- 42h — порт устройства не был заблокирован;
- 43h — ошибка блокировки (некорректный адрес порта).

Использование EPP BIOS при работе с принтерами

К сожалению, дополнительный набор функций BIOS, который мы рассмотрим ниже, появился со значительным опозданием и до сих пор не поддерживается многими изготовителями системных плат. Кроме того, этот набор, как явствует из его названия, ориентирован на использование режима EPP, который практически не поддерживается изготовителями принтеров.

При работе с принтерами реальную пользу могут принести только головная функция, позволяющая проверить наличие EPP BIOS и получить вектор EPP, и функция для установки режима работы контроллера параллельного порта. Функция установки режима представляет особый интерес, так как позволяет задать нужный режим работы контроллера прямо из прикладной программы (до появления

EPP BIOS операцию выбора режима можно было осуществить только в процессе начальной загрузки компьютера, через BIOS Setup).

Все остальные функции EPP BIOS при работе с принтером могут пригодиться только в том случае, если принтер подключен к компьютеру через автоматический мультиплексор. В России, однако, такие мультиплексоры используются крайне редко вследствие относительного дефицита периферийного оборудования (на один системный блок редко приходится более одного принтера, а со сканером и внешним дисководом принтер обычно объединяют в цепочку, без мультиплексора).

В листинге 7.2 приведен пример программы, которая использует функцию установки режима работы EPP BIOS для переключения контроллера параллельного порта в режим ECP.

Листинг 7.2. Программа для проверки наличия EPP BIOS и переключения порта LPT1 в режим ECP

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

DATASEG
; Вектор точки входа EPP BIOS
label EPP_Vector DWORD
EPP_Offset      DW ?
EPP_Segment     DW ?
; Конфигурация порта EPP
InterruptLevel  DB ?
BIOS_Revision   DB ?
IO_Capabilities DB ?
IO_BaseAddress  DW ?
label EPP_Manufacturer DWORD
Manuf_Offset    DW ?
Manuf_Segment   DW ?
; Режим работы параллельного порта
OperationMode   DB ?
; Текстовые сообщения
AnyK DB YELLOW,24,29,"Нажмите любую клавишу",0
Txt1 DB LIGHTCYAN,0,28,"ТЕСТИРОВАНИЕ EPP BIDS",0
Txt2 DB 2,23,"Вектор EPP BIOS:     :",0
```

продолжение ⇒

Листинг 7.2 (*продолжение*)

```
    DB 3,8,"Номер используемого прерывания:",0
    DB 4,17,"Номер версии EPP BIOS:",0
    DB 5,14,"Возможности ввода-вывода:",0
    DB 6,3,"Базовый адрес блока регистров порта:",0
    DB 8,0,"Разработчик BIOS:",0
ECPY DB LIGHTCYAN,12,26,"Порт переключен в режим ECP",0
ECPN DB LIGHTRED,12,22,"Режим ECP не поддерживается портом",0
Err1 DB 12,22,"EPP BIOS не поддерживается системой",0
ENDS

SEGMENT sseg para stack 'STACK'
        DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC EPP_BIOS_Test
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        mov     SI,offset Txt1
        call    ShowColorString
        mov     SI,offset AnyK
        call    ShowColorString
; Инициализация EPP
        mov     AH,2
        mov     DX,0
        mov     AL,0
        mov     CH,'E'
        mov     BX,"PP"
        int     17h
; Проверка наличия EPP BIOS
        or      AH,AH
        jnz     @@No_EPP
        cmp     AL,'E'
        jne     @@No_EPP
        cmp     CX,"PP"
        jne     @@No_EPP
; Сохранение вектора точки входа EPP BIOS
```

```
        mov     [EPP_Offset],BX
        mov     [EPP_Segment],DX
; Вывести наименования параметров
        mov     [TextColorAndBackground],LIGHTGREEN
        MShowText 6,Txt2
        mov     [TextColorAndBackground],WHITE
; Вывести вектор точки входа EPP BIOS на экран
        MShowHexWord 2,40,[EPP_Segment]
        MShowHexWord 2,45,[EPP_Offset]
; Определить конфигурацию порта EPP
        push    ES
        mov     AH,0
        mov     DL,0
        call    [Epp_Vector]
; Сохранить параметры настройки порта
        mov     [InterruptLevel],AL
        mov     [BIOS_Revision],BH
        mov     [IO_Capabilities],BL
        mov     [IO_BaseAddress],CX
        mov     [Manuf_Offset],DI
        mov     AX,ES
        mov     [Manuf_Segment],AX
        pop     ES
; Вывести значения параметров
        MShowHexByte 3,40,[InterruptLevel]
        MShowHexByte 4,40,[BIOS_Revision]
        MShowBinByte 5,40,[IO_Capabilities]
        MShowHexWord 6,40,[IO_BaseAddress]
        ; Вывести имя разработчика BIOS
        pusha
        push    DS
        push    ES
        mov     AX,0B800h
        mov     ES,AX
        ; Задать позицию строки в видеопамяти
        mov     DI,160*8 + 18*2
        ; Использовать цвет, заданный по умолчанию
        mov     AH,[TextColorAndBackground]
        ; Установить указатель на начало ASCIIZ-строки
        mov     SI,[Manuf_Offset]
        mov     BX,[Manuf_Segment]
        mov     DS,BX
        mov     CX,62 ;ограничитель длины строки
@@NextChar:
        lodsb
        and     AL,AL ;конец строки?
        jz      @@Zero
        stosw
        loop    @@NextChar
@@Zero: pop     ES
```

продолжение ⇗

Листинг 7.2 (*продолжение*)

```
        pop     DS
        popa

; Переключить порт в режим ECP
        mov     AH,1
        mov     DL,0
        mov     AL,1000b
        call    [Epp_Vector]
        ; Операция выполнена?
        cmp     AH,0
        jne     @@ECP_Not_Supported
        MShowColorString ECPY ;режим ECP установлен
        jmp     @@End
@@ECP_Not_Supported:
        MShowColorString ECPN ;режии ECP не установлен

@@End:  call    GetChar
; Переустановить текстовый режим
        mov     ax,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h

; Сообщения об ошибках
@@No_EPP:
        MFatalError Errl ; отсутствует EPP BIOS
ENDP EPP_BIOS_Test
ENDS

; Подключить процедуры вывода данных на экран
include "listl_02.inc"

END
```

Непосредственная работа с регистрами параллельного порта в режиме SPP

Если функции BIOS по каким-то причинам использовать невозможно (например, при работе в защищенном режиме), то приходится работать с принтером напрямую через регистры параллельного порта, к которому он подключен. Любая современная системная (материнская) плата персонального компьютера содержит в своем составе контроллер устройства LPT1 и имеет соответствующий разъем

для подключения принтера. В большинстве случаев этого порта достаточно, но при необходимости можно установить дополнительный контроллер для работы с устройством LPT2 в один из разъемов расширения. По традиции принято, что устройство LPT1 может вырабатывать запрос на прерывание IRQ7 (Int 0Fh), а устройство LPT2 – IRQ5 (Int 0Dh), однако BIOS SETUP современных системных плат позволяет запретить эти прерывания или переопределить их на другие.

Со времени разработки компьютеров серии IBM PC конструкция параллельного порта претерпела существенные изменения, однако, с целью обеспечения совместимости со старым программным обеспечением, после включения питания компьютера и выполнения процесса начальной загрузки контроллер порта будет настроен на режим совместимости (Compatibility Mode) стандартного порта.

Стандартный параллельный порт (Standard Parallel Port, сокращенно SPP) имеет три регистра, расположенные в пространстве ввода-вывода последовательно, друг за другом. Адреса регистров для портов LPT1 и LPT2 перечислены в табл. 7.1.

Таблица 7.1. Регистры интерфейса параллельной передачи данных

Адрес регистра		Назначение регистра
Устройство LPT1	**Устройство LPT2**	
378h	278h	Регистр данных
379h	279h	Регистр состояния
37Ah	27Ah	Регистр управления

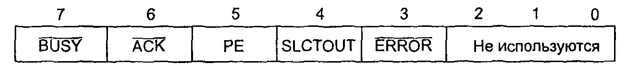
Рассмотрим указанные регистры более подробно.

Регистр данных (порт 378h для LPT1, порт 278h для LPT2) доступен для выполнения операций записи и считывания, применяется в основном для вывода информации на принтер.

Регистр состояния (порт 379h для LPT1, порт 279h для LPT2), формат которого показан на рис. 7.2, доступен только для чтения. Назначение разрядов регистра следующее:

- биты 0–2 – не используются, установлены в 0;
- бит 3 – признак ошибки ввода-вывода (0 – ошибка, 1 – нет ошибки);
- бит 4 – признак выбора принтера (0 – принтер в автономном режиме, 1 – принтер в режиме подключения);
- бит 5 – контроль наличия бумаги (0 – бумага вставлена, 1 – нет бумаги);

- бит 6 — подтверждение приема (0 — подтверждение приема символа, 1 — обычное состояние);
- бит 7 — признак занятости принтера (0 — принтер занят, 1 — принтер свободен).

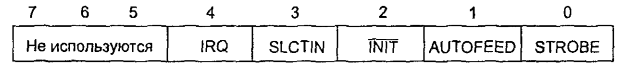


7	6	5	4	3	2 1 0
$\overline{\text{BUSY}}$	$\overline{\text{ACK}}$	PE	SLCTOUT	$\overline{\text{ERROR}}$	Не используются

Рис. 7.2. Формат регистра состояния принтера

Регистр управления принтером (порт 37Ah для LPT1, порт 27Ah для LPT2), формат которого показан на рис. 7.3, доступен для записи и считывания. Назначение разрядов регистра управления:

- бит 0 — строб данных (0 — обычное состояние, 1 — выполнить вывод байта данных на принтер);
- бит 1 — автоматический перевод строки после возврата каретки;
- бит 2 — сброс принтера (0 — выполнить сброс принтера, 1 — обычное состояние);
- бит 3 — выбор принтера (0 — отмена выбора принтера, 1 — обычное состояние);
- бит 4 — разрешение прерывания от принтера (0 — прерывание запрещено, 1 — разрешено);
- биты 5–7 — не используются, установлены в 0.



7 6 5	4	3	2	1	0
Не используются	IRQ	SLCTIN	$\overline{\text{INIT}}$	AUTOFEED	STROBE

Рис. 7.3. Формат регистра управления принтером

Процесс передачи байта данных

Для передачи одного байта данных необходимо выполнить следующие операции:

1. поместить передаваемый байт в регистр данных;
2. проверить готовность принтера к приему данных, прочитав регистр состояния и выделив старший разряд. Если значение старшего разряда равно 1, можно приступать к передаче байта; если значение равно 0 — снова опросить регистр состояния. Количество повторений цикла опроса должно быть ограничено по счетчику повторений или по времени, чтобы избежать зависания

программы (если интервал ожидания готовности исчерпан, выдается сообщение об ошибке);

3. загрузить в регистр управления значение 0Dh (установка в 1 младшего разряда регистра управления формирует сигнал стробирования данных);
4. ожидать поступления сигнала подтверждения приема данных (когда данные приняты, бит 6 регистра состояния сбрасывается в 0). Цикл ожидания должен быть ограничен по времени;
5. загрузить в регистр управления значение 0Ch (в результате чего сигнал стробирования снимается).

Например, для параллельного порта LPT1, имеющего базовый адрес 378h, программный код передачи байта будет выглядеть следующим образом:

```
; Загрузить символ в регистр данных
        mov     DX,378h ;DX адресует регистр данных
        mov     AL,AH
        out     DX,AL
; Проверить состояние принтера
        inc     DX      ;DX адресует регистр состояния
        xor     CX,CX   ;выполнить 65536 циклов опроса
@@Busy: in      AL,DX
        test    AL,80h  ;проверить бит готовности
        jnz     @@Str1
        loop    @@Busy
        ; Интервал ожидания исчерпан, ошибка передачи
        jmp     @@Error
; Подать сигнал стробирования
@@Str1: inc     DX      ;DX адресует регистр управления
        mov     AL,0Dh
        out     DX,AL
; Ожидать поступление сигнала подтверждения
        dec     DX      ;DX адресует регистр состояния
        mov     CX,1000
@@Wait: in      AL,DX
        test    AL,40h  ;проверить бит подтверждения
        jnz     @@Str0
        loop    @@Wait
; Снять сигнал стробирования
@@Str0: inc     DX      ;DX адресует регистр управления
        mov     AL,0Ch
        out     DX,AL
```

ПРИМЕЧАНИЕ

Следует отметить, что непосредственная работа с регистрами порта в режиме SPP не дает заметного выигрыша в скорости передачи данных по сравнению с функциями BIOS.

Работа контроллера параллельного порта в режиме ECP

Спецификация ECP была разработана фирмами Microsoft и Hewlett-Packard. Она предусматривает введение в контроллер параллельного порта дополнительного блока регистров, изменение назначения стандартных регистров и использование специальных протоколов, увеличивающих скорость передачи данных более чем на порядок (со 1-50 Кбайт/с в режиме SPP до 2–5 Мбайт/с в режиме ECP).

Регистры контроллера параллельного порта в режиме ECP

Перечень регистров параллельного порта для режима ECP приведен в табл. 7.2. В графе «Доступ» используются следующие обозначения:

- R/W — регистр доступен для чтения и записи данных,
- R — регистр доступен только для чтения.

Таблица 7.2. Регистры ECP

Мнемоника	Смещение	Доступ	Режимы ECP	Наименование
Data	000h	R/W	000, 001	Регистр данных SPP
EcpAFifo	000h	R/W	011	Регистр очереди адресов ECP
Dsr	001h	R	Все	Регистр состояния
Dcr	002h	R/W	Все	Регистр управления
CFifo	400h	R/W	010	Регистр очереди данных SPP
EcpDFifo	400h	R/W	011	Регистр очереди данных ECP
TFifo	400h	R/W	110	Регистр тестирования очереди
CnfgA	400h	R	111	Конфигурационный регистр A
CnfgB	401h	R/W	111	Конфигурационный регистр B
Ecr	402h	R/W	Все	Дополнительный регистр управления

Регистр данных SPP (Parallel port Data Register, сокращенно data) имеет смещение 000h и используется в режимах 000 и 001, обеспечивающих совместимость со стандартным режимом работы параллельного порта — SPP.

Регистр очереди адресов ЕСР (ECP Address FIFO, сокращенно ecpAFifo) имеет смещение 000h и используется только в режиме 011. Байт данных, записанный в регистр ecpAFifo, помещается в очередь FIFO и трактуется либо как адрес, либо как код повторения RLE. Регистр ecpAFifo имеет следующую структуру:

- биты 0–6 — адрес или код повторения RLE;
- бит 7 — тип данных, записанных в битах 0–6 (0 — код RLE, 1 — адрес ЕСР).

Регистр состояния (Device Status Register, сокращенно dsr), формат которого показан на рис. 7.4, доступен только для чтения и имеет смещение 001h. Его структура идентична структуре регистра состояния стандартного параллельного порта (только для разрядов используются иные мнемонические обозначения):

- биты 0–2 зарезервированы, и их значение при считывании не стандартизировано, поэтому их следует игнорировать;
- бит 3 — признак отсутствия ошибки (в стандарте ЕСР обозначается как nFault);
- бит 4 — признак выбора принтера (в стандарте ЕСР обозначается как Select);
- бит 5 — контроль наличия бумаги (в стандарте ЕСР обозначается как PError);
- бит 6 — подтверждение приема (в стандарте ЕСР обозначается как nAck);
- бит 7 — признак занятости принтера (в стандарте ЕСР обозначается как nBusy).

7	6	5	4	3	2	1	0
nBusy	nAck	PError	Select	nFault	Не используются		

Рис. 7.4. Формат регистра состояния dsr

Регистр управления (Device Control Register, сокращенно dcr), формат которого показан на рис. 7.5, имеет смещение 002h и доступен как чтения, так и для записи. Его структура аналогична структуре

регистра управления стандартного параллельного порта (за исключением пятого бита):

- бит 0 — строб данных (в стандарте ECP обозначается как strobe);
- бит 1 — автоматический перевод строки (в стандарте ECP обозначается как autofd);
- бит 2 — сброс принтера (в стандарте ECP обозначается как nInit);
- бит 3 — выбор принтера (в стандарте ECP обозначается как SelectIn);
- бит 4 — разрешение прерывания при переходе сигнала nAsk из 0 в 1 (в стандарте ECP обозначается как ackIntEn);
- бит 5 — направление передачи данных (в стандарте ECP обозначается как direction);
- биты 6 и 7 зарезервированы.

Бит Direction регистра управления задает направление передачи данных:

- 0 — вывод данных, выходные усилители на линиях данных активны;
- 1 — вывод данных, выходные усилители на линиях данных отключены.

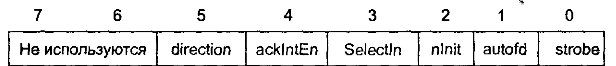


Рис. 7.5. Формат регистра управления dcr

Переключение бита direction в состояние 1 возможно только в режиме 001. В режимах 000 и 010 возможен только вывод данных, и значение бита direction игнорируется.

Регистр очереди данных SPP (Parallel Port Data FIFO, сокращенно cFifo) имеет смещение 400h и используется только в режиме 010. Регистр доступен как чтения, так и для записи, однако очередь работает только на вывод данных. Запись данных в регистр может осуществляться процессором или контроллером DMA. Следует отметить, что при работе со встроенным параллельным портом чипсеты системных плат допускают использование только 8-разрядного режима DMA.

Регистр очереди данных ECP (ECP Data FIFO, сокращенно ecpDFifo) имеет смещение 400h и используется только в режиме 011. Регистр доступен как чтения, так и для записи. Операции чтения и записи

данных могут выполняться процессором или контроллером DMA (разрешен только 8-разрядный режим). Перед началом передачи данных бит direction регистра управления должен быть сброшен в 0, а перед началом приема — установлен в 1.

Регистр тестирования очереди (Test FIFO Mode, сокращенно tFifo) имеет смещение 400h и используется только в режиме 110. Регистр доступен как чтения, так и для записи. Операции чтения и записи данных могут выполняться процессором или контроллером DMA (разрешен только 8-разрядный режим).

Конфигурационный регистр A (Configuration Register A, сокращенно cnfgA) имеет смещение 400h, используется только в режиме 111, доступен только для чтения и содержит константу 10h (встроенный параллельный порт системной платы имитирует работу 8-разрядной шины ISA).

Конфигурационный регистр B (Configuration Register B, сокращенно cnfgB) имеет смещение 401h, используется только в режиме 111 и доступен только для чтения. Разряды регистра cnfgB имеют следующее назначение:

- биты 0–5 зарезервированы (содержат нули);
- бит 6 — признак наличия конфликта, связанного с использованием линии IRQ ISA (данный бит доступен только для чтения);
- бит 7 — управление аппаратным сжатием данных (у встроенного параллельного порта системной платы бит 7 доступен только для чтения и всегда сброшен в ноль — аппаратная компрессия не поддерживается).

7–5	4	3	2	1	0
Код режима работы	nErrIntEn	dmaEn	serviceIntr	full	empty

Рис. 7.6. Формат дополнительного регистра управления ecr

Дополнительный регистр управления (Extended Control Register, сокращенно ecr), формат которого показан на рис. 7.6, имеет смещение 402h и доступен для чтения и записи во всех режимах работы контроллера ECP. Разряды регистра ecr имеют следующее назначение:

- бит 0 (empty) — признак освобождения очереди (0 — очередь пуста, 1 — очередь содержит по крайней мере один байт данных);
- бит 1 (full) — признак отсутствия свободного места в очереди (0 — в очереди есть место по крайней мере для одного байта данных; 1 — очередь заполнена);

- бит 2 (serviceIntr) — блокировка служебных прерываний (0 — прерывания разрешены, 1 — запрещено использование DMA и обслуживание прерываний);
- бит 3 (dmaEn) — управление режимом DMA (0 — использование DMA запрещено, 1 — использование DMA разрешено при условии serviceIntr = 0);
- бит 4 (nErrIntEn) — блокировка обслуживания прерывания по сигналу nFault (0 — разрешено прерывание при переходе сигнала nFault с высокого уровня на низкий, 1 — прерывание от nFault запрещено);
- биты 5–7 — код режима работы контроллера ECP (табл. 7.3).

Таблица 7.3. Режимы работы контроллера ECP

Код	Режим работы контроллера
000	Режим SPP. Контроллер передает данные в стандартном режиме параллельного порта, бит direction регистра управления игнорируется, очередь данных не используется (очищается)
001	Режим параллельного порта PS/2. Кроме передачи данных в стандартном режиме, возможно осуществление приема информации с линий данных, когда бит direction регистра управления установлен в 1
010	Режим параллельного порта с очередью данных. Контроллер передает данные в стандартном режиме параллельного порта, однако, в отличие от режима 000, данные должны записываться процессором или контроллером DMA не в регистр data, а в регистр cFifo
011	Режим ECP. При выводе данных (бит direction регистра управления сброшен в 0) информация, записываемая в регистры ecpDFifo и ecpAFifo, заносится в общую очередь и передается автоматически под управлением контроллера ECP. При выполнении чтения (бит direction установлен в 1) информация из очереди данных считывается через регистр ecpDFifo
100	Режим EPP
110	Режим тестирования очереди данных. Информация может записываться в очередь или считываться из нее, но на выходные линии данных параллельного порта она не выдается
111	Режим конфигурирования контроллера. В этом режиме регистры cnfgA и cnfgB доступны для чтения и записи данных (у встроенного порта системной платы — только для чтения)

Управление работой контроллера ECP

Для управления работой контроллера могут использоваться сигналы прерываний.

Контроллер ECP генерирует прерывания в следующих случаях:

- serviceIntr = 0, dmaEn = 1, завершена прямая передача данных (счетчик DMA достиг конечного значения);
- serviceIntr = 0, dmaEn = 0, direction = 0 и имеется свободное место в очереди данных;
- serviceIntr = 0, dmaEn = 0, direction = 1 и в очереди имеется по крайней мере один байт данных;
- nErrIntEn = 0 и сигнал nFault переключился с высокого уровня на низкий;
- при переключении бита nErrIntEn из 1 в 0, если сигнал nFault имеет на низкий уровень;
- ackIntEn = 1 и сигнал nAsk переходит из 0 в 1.

Переключение режимов работы контроллера ECP осуществляется программным обеспечением через дополнительный регистр управления ecr. При установке направления передачи данных используется также бит direction регистра управления dcr.

Из режимов 000 и 001 контроллер можно сразу переключить в любой другой режим. Если контроллер находится в одном из режимов 010–111, то его можно переключить только в режим 000 или режим 001; для переключения в другие режимы необходимо вначале переключиться в режим 000 или 001. Переключение бита direction в регистре управления может производиться только в режиме 001. Перед переключением режима необходимо убедиться, что в очереди FIFO нет данных (очередь пуста).

Перед началом использования любого режима передачи данных, отличного от стандартного режима SPP, программное обеспечение должно выполнить процедуру переговоров (negotiation) с периферийным устройством для согласования режима работы. Согласование осуществляется в соответствии со стандартом IEEE 1284.1 [61].

ВНИМАНИЕ

Переговоры с периферийным устройством могут выполняться только в режимах 000 и 001 контроллера ECP.

После завершения процедуры согласования режима необходимо инициализировать регистр управления dcr, загрузив в него код 0Ch: strobe = 0, autofd = 0, nInit = 1, SelectIn = 1, ackIntEn = 0, direction = 0.

Далее производится установка режима ECP путем загрузки кода 74h в дополнительный регистр управления ecr.

Процедура переговоров

Для того чтобы между контроллером параллельного порта и периферийным устройством была возможна передача данных, режимы работы контроллера и устройства должны быть согласованы друг с другом.

Стандарт IEEE 1284.1 предусматривает определенный порядок переключения периферийных устройств (в том числе — принтеров) из стандартного режима SPP в другие режимы передачи данных, а также порядок возврата в стандартный режим передачи.

Процедуры переговоров для разных режимов различаются между собой. Ниже мы будем рассматривать только процедуру переключения устройства в режим ECP и процедуру возврата в режим SPP из режима ECP.

Стандарт IEEE 1284.1 рассматривает различные протоколы передачи данных с точки зрения сигналов на линиях связи, но для программиста такой подход неудобен, так как некоторые сигналы представлены в регистрах контроллера порта в инверсной форме. Во избежание путаницы с обозначением сигналов на линиях кабеля и в регистрах контроллера процедуру переговоров будем рассматривать с точки зрения программиста, работающего с регистрами порта.

Для того чтобы переключить периферийное устройство в режим ECP, необходимо выполнить перечисленные ниже операции.

1. Установить режим передачи SPP, загрузив в дополнительный регистр управления код 14h (прерывания блокированы, использование DMA запрещено, прерывание по сигналу nFault запрещено, режим работы — SPP).
2. Загрузить в регистр данных код режима ECP — значение 10h.
3. Загрузить в регистр управления код 0Ch (strobe = 0, autofd = 0, nInit = 1, SelectIn = 1).
4. Вставить задержку на один тик системного таймера (0,05 с).
5. Проверить готовность принтера к работе (биты Select и nBusy в регистре состояния должны быть установлены в 1).
6. Начать фазу переговоров, загрузив в регистр управления код 06h (strobe = 0, autofd = 1, nInit = 1, SelectIn = 0).
7. Вставить задержку на один тик системного таймера.
8. Проверить наличие в регистре состояния следующей комбинации сигналов: nFault = 1, Select = 1, Perror = 1, nAsk = 0 (любая

другая комбинация значений указывает на то, что принтер не соответствует стандарту IEEE 1284.1).

9. Загрузить в регистр управления код 07h (strobe = 1, autofd = 1, nInit = 1, SelectIn = 0).
10. Вставить задержку на одну микросекунду.
11. Загрузить в регистр управления код 04h (strobe = 0, autofd = 0, nInit = 1, SelectIn = 0).
12. Вставить задержку на один тик системного таймера.
13. Проверить наличие в регистре состояния следующей комбинации сигналов: Select = 1, Perror = 0, nAsk = 1 (любая другая комбинация значений указывает на то, что принтер не поддерживает режим ECP).
14. Начать фазу установки, загрузив в регистр управления код 06h (strobe = 0, autofd = 1, nInit = 1, SelectIn = 0).
15. Вставить задержку на один тик системного таймера.
16. Проверить значение бита PError в регистре состояния (если он не установлен в 1, произошел сбой).
17. Установить для контроллера порта режим ECP, загрузив в дополнительный регистр управления код 74h.
18. Установить нулевой адрес канала, загрузив в регистр данных значение 0.

Если все перечисленные операции выполнены успешно, принтер готов к приему данных в режиме ECP.

ПРИМЕЧАНИЕ

После выполнения фазы переговоров для устройства по умолчанию устанавливается нулевой адрес канала ECP. По этому адресу выполняется передача данных на принтер, а другие адреса обычно не используются.

После того как передача данных завершена, нужно выйти из режима ECP в стандартный режим (процесс возврата в стандартный режим именуется в документации завершающей фазой).

Для выхода из режима ECP требуется выполнить указанные ниже операции.

1. Установить для контроллера порта режим SPP, загрузив в дополнительный регистр управления код 14h.
2. Загрузить в регистр управления код 0Ch (strobe = 0, autofd = 0, nInit = 1, SelectIn = 1).

3. Вставить задержку на один тик системного таймера.
4. Проверить наличие в регистре состояния следующей комбинации сигналов: nFault = 1, Select = 0, nAsk = 0, nBusy = 0 (любая другая комбинация означает сбой в работе принтера).
5. Загрузить в регистр управления код 0Eh (strobe = 0, autofd = 1, nInit = 1, SelectIn = 1).
6. Вставить задержку на один тик системного таймера.
7. Проверить значение бита nAsk в регистре состояния (если он не установлен в 1, произошел сбой).
8. Загрузить в регистр управления код 0Ch (strobe = 0, autofd = 0, nInit = 1, SelectIn = 1).

Передача данных в режиме ECP

Существует три способа передачи данных в режиме ECP:

- программно-управляемая передача данных;
- передача данных по прерываниям;
- передача данных в режиме DMA.

Режим программно-управляемой передачи самый простой и самый неэффективный, так как требует постоянного контроля состояния очереди данных. Все процессорное время в этом случае тратится на управление передачей, причем большая часть расходуется на опрос регистра ecr.

Возможно несколько вариантов реализации программной передачи.

1. Процессор ожидает полного освобождения очереди и загружает в нее один байт данных.
2. Процессор ожидает появления свободного места в очереди и загружает в нее байты данных, пока очередь не будет заполнена.
3. Процессор ожидает полного освобождения очереди и загружает в нее сразу 16 байт данных.

Режим передачи по прерываниям позволяет освободить процессор от необходимости постоянно контролировать состояние очереди данных. Чтобы начать передачу данных по прерываниям, нужно настроить соответствующий порту вектор прерывания на обработчик прерывания и сбросить в 0 бит блокировки служебных прерываний serviceIntr в регистре ecr. В режиме передачи прерывание генерируется в том случае, когда в очереди появляется свободное место; в режиме приема прерывание вырабатывается, когда в очереди име-

ется хотя бы один байт. Обработчик прерывания должен знать, в каком режиме работы (передачи или приема) находится порт в данный момент, и в соответствии с этим либо загружать байт в очередь, либо считывать байт из очереди.

Использование DMA – самый эффективный, но самый сложный (с точки зрения настройки) способ передачи данных.

Подготовка к передаче в режиме DMA начинается с установки направления передачи. Далее следует настроить третий канал контроллера DMA, загрузив в него адрес буфера данных в оперативной памяти и количество передаваемых байтов. Затем нужно установить вектор прерывания от параллельного порта на обработчик прерывания по завершению передачи в режиме DMA. После этого нужно разрешить служебные прерывания и использование DMA, для чего следует сбросить в 0 бит блокировки служебных прерываний serviceIntr и установить в 1 бит управления режимом DMA dmaEn в регистре ecr. Процесс передачи данных будет происходить без участия центрального процессора; когда передача данных будет завершена, следует запретить служебные прерывания и режим DMA.

Переключение направления передачи данных

Переключение направления передачи данных производится путем изменения значения бита direction в регистре управления. Переключение направления передачи, однако, разрешено только в режиме совместимости, поэтому контроллер должен быть временно переключен в режим 001. В этом режиме контроллер должен выполнить процедуру согласования направления передачи данных.

Процесс передачи данных от компьютера к периферийному устройству в спецификации ЕСР называется фазой прямой передачи данных (Forward Phase), а процесс передачи от периферийного устройства к компьютеру – фазой обратной передачи (Reverse Phase). После установки режима ЕСР контроллер параллельного порта и периферийное устройство настроены на прямую передачу данных.

Для переключения направления передачи с прямого на инверсное нужно выполнить перечисленные ниже операции.

1. Загрузить в регистр управления код 06h (strobe = 0, autofd = 1, nInit = 1, SelectIn = 0).
2. Вставить задержку на одну микросекунду.

3. Загрузить в регистр управления код 02h (strobe = 0, autofd = 1, nInit = 0, SelectIn = 0).
4. Вставить задержку на один тик системного таймера.
5. Проверить значение бита PError в регистре состояния (если он не сброшен в 0, произошел сбой).

Для переключения направления передачи с инверсного на прямое нужно выполнить указанные далее операции.

1. Загрузить в регистр управления код 06h (strobe = 0, autofd = 1, nInit = 1, SelectIn = 0).
2. Вставить задержку на один тик системного таймера.
3. Проверить значение бита PError в регистре состояния (если он не установлен в 1, произошел сбой).

ПРИМЕЧАНИЕ

Использование инверсного режима связано с определеным риском и требует неукоснительного соблюдения правил перключения между прямым и инверсным режимом, указанных в стандарте IEEE 1284.1 и спецификации ECP. При нарушении порядка перключения параллельный порт компьютера и порт принтера могут одновременно оказаться в режиме передачи, что приведет к повреждению выходных усилителей портов.

Пример программы, которая выполняет переключение в режим ECP, передает на принтер строку символов, состоящую из заглавных латинских букв от A до Z, а затем выполняет возврат в режим SPP, приведен в листинге 7.3.

Листинг 7.3. Программа для проверки способности принтера работать в режиме ECP

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

DATASEG
; Текстовые сообщения
Txtl DB LIGHTCYAN,0,19
    DB "ПЕЧАТЬ ТЕКСТА С ИСПОЛЬЗОВАНИЕМ РЕЖИМА ECP",0
```

```
AnyK DB YELLOW,24,29,"Нажмите любую клавишу",0
WPrn DB YELLOW,12,26,"Ждите завершения печати ...",0
Err1 DB 12,25,"Порт находится не в режиме ECP",0
Err2 DB 12,27,"Принтер не готов к работе",0
Err3 DB 12,21,"Режим ECP не поддерживается принтером",0
ENDS

SEGMENT sseg para stack 'STACK'
        DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC Test_ECP_Mode
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Проверить режим работы порта
        mov     DX,378h+402h
        in      AL,DX
        cmp     AL,0FFh
        je      @@Err1
; Вывести текстовые сообщения на экран
        mov     SI,offset Txt1
        call    ShowColorString
        mov     SI,offset WPrn
        call    ShowColorString

; ПОДГОТОВИТЕЛЬНАЯ ФАЗА
; Установить для порта режим SPP
        mov     DX,378h+402h
        mov     AL,00010100b
        out     DX,AL
; Загрузить код режима ECP в регистр данных
        mov     DX,378h
        mov     AL,00010000b
        out     DX,AL
; Установить strobe в 0, AutoFeed в 0, SelectIn в 1
        mov     DX,378h+2
        mov     AL,00001100b
        out     DX,AL
```

продолжение ⇩

Листинг 7.3 (*продолжение*)

```
        call    Wait05s ;задержка на 0,05 с
; Принтер готов к работе?
        mov     DX,378h+1
        in      AL,DX
        and     AL,10010000b
        cmp     AL,10010000b
        jne     @@Err2

; ФАЗА ПЕРЕГОВОРОВ
; Установить strobe в 0, AutoFeed в 1, SelectIn в 0
        mov     DX,378h+2
        mov     AL,00000110b
        out     DX,AL
        call    Wait05s ;задержка на 0,05 с
; Принтер соответствует стандарту IEEE 1284?
        mov     DX,378h+1
        in      AL,DX
        and     AL,01111000b
        cmp     AL,00111000b
        jne     @@Err3
; Установить strobe в 1, AutoFeed в 1, SelectIn в 0
        mov     DX,378h+2
        mov     AL,00000111b
        out     DX,AL
        call    Wait1us ;задержка на 1 мкс
; Установить strobe в 0, AutoFeed в 0, SelectIn в 0
        mov     DX,378h+2
        mov     AL,00000100b
        out     DX,AL
        call    Wait05s ;задержка на 0,05 с
; Принтер поддерживает режим ECP?
        mov     DX,378h+1
        in      AL,DX
        and     AL,01110000b
        cmp     AL,01010000b
        jne     @@Err3

; ФАЗА УСТАНОВКИ
; Установить strobe в 0, AutoFeed в 1, SelectIn в 0
        mov     DX,378h+2
        mov     AL,00000110b
        out     DX,AL
        call    Wait05s ;задержка на 0,05 с
; Бит PError установлен?
        mov     DX,378h+1
        in      AL,DX
        test    AL,00100000b
        jz      @@Err3
; Установить для порта режим ECP
```

```
        mov     DX,378h+402h
        mov     AL,01110100b
        out     DX,AL
; Установка нулевого адреса канала
        mov     DX,378h
        mov     AL,0
        out     DX,AL

; ПЕРЕДАЧА ДАННЫХ НА ПРИНТЕР
; Печать строки символов (от A до Z)
        mov     AL,'A'
@@OutNextChar:
        call    ECP_Out
        inc     AL
        cmp     AL,'Z'
        jbe     @@OutNextChar
        mov     AL,0Dh ;возврат каретки
        call    ECP_Out
        mov     AL,0Ah ;перевод строки
        call    ECP_Out
        mov     AL,0Ch ;перевод формата
        call    ECP_Out

; Вывести текстовые сообщения на экран
        call    ClearScreen
        mov     SI,offset Txt1
        call    ShowColorString
        mov     SI,offset AnyK
        call    ShowColorString
; Ожидать нажатия клавиши
        call    GetChar

; ЗАВЕРШАЮЩАЯ ФАЗА
; Установить для порта режим SPP
        mov     DX,378h+402h
        mov     AL,00010100b
        out     DX,AL
; Установить SelectIn в 1, AutoFeed в 0, strobe в 0
        mov     DX,378h+2
        mov     AL,00001100b
        out     DX,AL
        call    Wait05s ;задержка на 0,05 с
; В регистре состояния присутствует комбинация
; nFault=1, Select=0, nAck=0, nBusy=0?
        mov     DX,378h+1
        in      AL,DX
        and     AL,11011000b
        cmp     AL,00001000b
        jne     @@Err3
; Установить strobe в 0, AutoFeed в 1, SelectIn в 1
```

продолжение ⇨

Листинг 7.3 (*продолжение*)

```
        mov     DX,378h+2
        mov     AL,00001110b
        out     DX,AL
        call    Wait05s ;задержка на 0,05 с
; В регистре состояния бит nAck=1?
        mov     DX,378h+1
        in      AL,DX
        test    AL,01000000b
        jz      @@Err3
; Установить SelectIn в 1, AutoFeed в 0, strobe в 0
        mov     DX,378h+2
        mov     AL,00001100b
        out     DX,AL

; Переустановить текстовый режим
        mov     ax,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h

; Сообщения об ошибках
@@Err1: MFatalError Err1 ;порт не в режиме ECP
@@Err2: MFatalError Err2 ;принтер не готов к работе
@@Err3: MFatalError Err3 ;режим ECP не поддерживается
ENDP Test_ECP_Mode

;*************************************************
;* ПРОЦЕДУРА ДЛЯ ВЫВОДА БАЙТА ДАННЫХ В РЕЖИМЕ ECP *
;* Передаваемые параметры:                       *
;* AL - выводимый байт данных.                   *
;*************************************************
PROC ECP_Out near
        push    AX
        push    DX
        mov     AH,AL
; Ожидать очистки очереди данных ECP
        mov     DX,378h+402h
@@Wait_FIFO_Empty:
        in      AL,DX
        and     AL,00000001b
        jz      @@Wait_FIFO_Empty
; Загрузить байт данных в очередь
        sub     DX,2
        mov     AL,AH
        out     DX,AL
        pop     DX
        pop     AX
```

```
        ret
ENDP ECP_Out

;*****************************************
;* ЗАДЕРЖКА НА ОДИН ТИК СИСТЕМНОГО ТАЙМЕРА *
;*****************************************
PROC Wait05s near
        push    ES
        push    EAX
        mov     AX,0
        mov     ES,AX
        mov     EAX,[ES:046Ch]
        inc     EAX
@@Wait: cmp     EAX,[ES:046Ch]
        jae     @@Wait
        pop     EAX
        pop     ES
        ret
ENDP Wait05s

;*******************************
;* ЗАДЕРЖКА НА ОДНУ МИКРОСЕКУНДУ *
;*******************************
PROC Wait1us near
        push    CX
        mov     CX,1000
@@Wait: nop
        loop    @@Wait
        pop     CX
        ret
ENDP Wait1us
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"

END
```

ПРИМЕЧАНИЕ

Пример из листинга 7.3 пригоден только для принтеров, поддерживающих текстовый режим печати. Перед запуском программы контроллер порта должен быть настроен на режим ECP. Если системная плата поддерживает EPP BIOS, можно выполнить переключение контроллера при помощи программы, приведенной в листинге 7.2. Если EPP BIOS не поддерживается, нужно переключить контроллер в режим ECP в процессе начальной загрузки компьютера, с помощью BIOS SETUP. Проверьте также базовый адрес набора регистров порта — пример рассчитан на использование стандартного адреса (378h).

Виды растровой печати

В современных принтерах применяются два режима растровой печати: режим битового образа и растровый режим. Основное различие между этими режимами заключается в том, что в растровом режиме байты данных выводятся вдоль направления строки, а в режиме битового образа — поперек.

Режим битового образа происходит от матричных принтеров и обеспечивает только черно-белую печать [46, 47]. Печать байтов данных поперек строки растра продиктована формой печатающей головки. Головка движется слева направо и выводит за одну операцию сразу от 8 до 48 точек. Соответственно, для передачи информации об одной колонке точек требуется передать от одного до 6 байтов. Байты данных при этом нумеруются сверху вниз, слева направо, как показано на рис. 7.7. Старший разряд байта данных отображается сверху, младший — снизу.

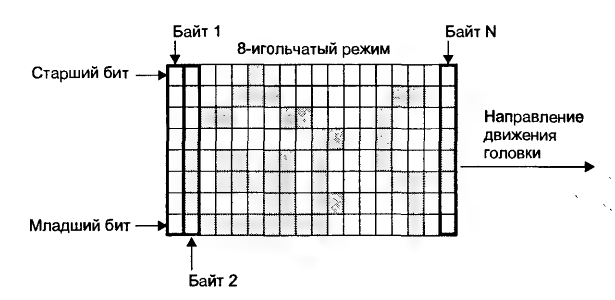
Растровый режим изначально был предназначен для лазерных принтеров, но в последнее время на него переведены также и новые модели струйных принтеров. Данный режим не зависит от количества сопел (иголок), он пригоден как для черно-белой, так и для цветной печати.

Порядок вывода информации на печать в монохромном (черно-белом) растровом режиме показан на рис. 7.8. Байты данных нумеруются слева направо, сверху вниз (как при выводе на экран монитора); каждый байт содержит информацию о восьми пикселах, причем старший разряд отображается слева, младший — справа.

Цветная печать предполагает наличие отдельной цветовой плоскости для каждого из основных цветов, формирующих изображение (бирюзового, пурпурного и желтого); возможно также наличие отдельной плоскости для черного цвета.

Каждая цветовая плоскость имеет структуру, аналогичную той, что изображена на рис. 7.8: каждой точке растра соответствует один бит плоскости.

Процедура передачи данных в цветном растровом режиме состоит из трех вложенных циклов: внешнего цикла по строкам растра, цикла по цветовым плоскостям и внутреннего цикла побайтной передачи строки цветовой плоскости. Таким образом, вначале поочередно передаются все цветовые компоненты первой строки растра, затем — второй и т. д. Порядок передачи цветовых компонентов определяется изготовителем принтера.



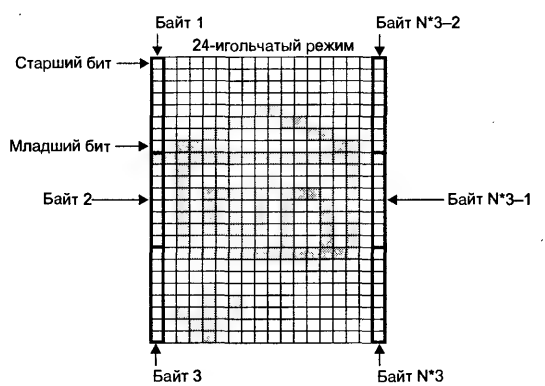


Рис. 7.7. Представление графической информации при печати в режиме битового образа

	Старший бит ... Младший бит			
Строка 1	Байт 1	Байт 2	...	Байт N
Строка 2	Байт N+1	Байт N+2	...	Байт 2N
...	...	...	...	...
Строка К	Байт(К–1)N+1	Байт(К–1)N+2	...	Байт KN+1

Рис. 7.8. Представление информации при печати в монохромном растровом режиме

Управление размещением графических изображений на странице

Перед началом выполнения процесса печати необходимо задать размеры ограничительных полей по краям страницы, определив четыре значения:

- отступ от верхнего края листа бумаги (Top margin);
- отступ от нижнего края (Bottom margin);
- отступ от левого края (Left margin);
- отступ от правого края (Right margin).

Область листа бумаги, ограниченная этими полями, называется областью печати: ни текст, ни графика не могут выходить за пределы этой области.

Обычно размеры полей являются одинаковыми для всех страниц печатаемого документа, поэтому задаются только один раз — перед началом печати первой страницы. Поля не могут быть уже технологических границ, задаваемых конструктивными особенностями конкретной модели принтера: тонер должен попадать только на бумажный лист, а не мимо листа на прижимной вал.

Технологические границы области печати зависят от формата бумажных носителей и задаются одной командой, определяющей тип носителя (например, лист бумаги формата A4, этикетка, почтовый конверт и т. д.).

После того, как задан формат носителя (физические размеры листа), можно приступать к определению границ области печати. В зависимости от используемого командного языка отступы задаются либо отдельными командами и в произвольном порядке, либо одновременно, при помощи одной команды. Встроенный микропроцессор принтера проверяет, чтобы заданные отступы были не уже технологических границ (в противном случае команда просто игнорируется)

Перед началом печати каждого отдельного изображения нужно задать его положение на странице, введя координаты левого верхнего угла изображения относительно левого верхнего угла области печати (координатная ось X при этом направлена слева направо, а ось Y — сверху вниз). Ширина изображения обычно задается в команде передачи графической строки (как ширина строки в пикселах), а высота определяется количеством растровых строк в изображении.

Набор команд Epson

Фирма Epson разработала для своих матричных принтеров набор команд ESC/P, фактически ставший впоследствии международным стандартом — фирмы, выпускающие матричные принтеры, в обязательном порядке включают в свои изделия поддержку набора команд ESC/P.

В набор ESC/P входят команды для печати в текстовом режиме и режиме битового образа. В усовершенствованный вариант этого командного языка, получивший название ESC/P2, были включены также команды для печати в растровом режиме. На основе ESC/P2 в свою очередь был разработан набор команд для струйных принтеров «Epson raster», специально ориентированный на использование растрового режима — команды для печати в текстовом режиме и режиме битового образа из него изъяты, зато добавлены новые растровые команды.

Группа команд общего назначения

Существует несколько универсальных команд, входящих во все командные языки и доступных в любых режимах печати. Эти команды стали стандартными, поскольку происходят от самых первых периферийных устройств, предназначенных для вывода текстовой информации, то есть от электрических пишущих машинок (см. табл. 1.1 в главе 1 «Работа с клавиатурой»).

В универсальную группу входят следующие команды:

- перевод строки (CR), код 0Dh;
- возврат каретки (LF), код 0Ah;
- перевод формата (FF), код 0Ch.

По команде «перевод строки» печатающее устройство перемещает лист бумаги (поперек направления печати) на расстояние, равное высоте одной строки (это расстояние регулируется при помощи специальной подгруппы команд принтера). При поступлении команды «возврат каретки» печатающее устройство устанавливает свой пишущий элемент в начало строки. Команда «перевод формата» позволяет извлечь из принтера отпечатанную страницу.

Кроме универсальных команд, в группу команд общего назначения, выполняемых всеми принтерами Epson, входят команда инициализации и команда установки абсолютной позиции по горизонтали.

По команде «Инициализаця» встроенный микропроцессор принтера переустанавливает все параметры печати в состояние, принятое по умолчанию. Команда инициализации имеет следующий формат:

<Esc>. '@'

ПРИМЕЧАНИЕ

Если команда состоит из нескольких байтов, то первым байтом в командной последовательности является символ Escape, которому соответствует шестнадцатеричный код 1Bh, поэтому многобайтные команды в литературе часто именуют Esc-последовательностями.

Команда «Установить абсолютное значения координаты по горизонтали» задает смещение позиции печати по горизонтали от левого края области печати. Команда имеет следующий формат:

<Esc>, '$', nL, nH

Параметр n в данной команде задает (в двоичном коде) абсолютную координату по X в единицах перемещения (nL — младший байт параметра, nH — старший байт).

Команды Epson для печати в режиме битового образа

Режим битового образа считается устаревшим, но до сих пор используется в матричных принтерах и некоторых моделях струйных принтеров Epson. Команды, используемые при печати в режиме битового образа, перечислены в табл. 7.4. Параметры команд, передаваемые в двоичном коде, обозначены в таблице прописными латинскими буквами.

Чтобы графическое изображение было сплошным, перед началом печати изображения необходимо установить межстрочное расстояние в дюймах равным высоте строки. Эта операция выполняется при помощи команд установки межстрочного расстояния.

Таблица 7.4. Команды Epson для печати в режиме битового образа

Название команды	Esc-последовательность
Установить межстрочное расстояние n/72 дюйма	<ESC> 'A' n
Установить межстрочное расстояние n/216 дюйма	<ESC> '3' n
Установить межстрочное расстояние n/360 дюйма	<ESC> '+' n
Напечатать графическую строку	<ESC> '*' m nL nH d1jdk

Для 9-игольчатых принтеров используется команда «Установить межстрочное расстояние n/72 дюйма» с параметром 8, которая воспроизводится следующей Esc-последовательностью:

<Esc>, 'A', 8

В шестнадцатеричном коде указанная последовательность будет выглядеть следующим образом: 1Bh, 41h, 08h.

Другая команда, дающая совершенно аналогичный результат — «Установить межстрочное расстояние n/216 дюйма» с параметром 24:

<Esc>, '3', 24

В шестнадцатеричном коде эта команда будет выглядеть так: 1Bh, 33h, 18h.

Для 24-игольчатых и струйных принтеров с той же целью нужно использовать команду «Установить межстрочное расстояние n/360 дюйма» с параметром 48:

<Esc>, '+', 48

В шестнадцатеричном коде эта команда имеет следующий вид: 1Bh, 28h, 30h.

Смещение изображения от левого края поля печати задается командой «Установить абсолютное значения координаты по горизонтали», которую следует подавать перед началом печати каждой строки изображения. Величина единицы перемещения (шаг перемещения) для режима битового образа равна 1/60 дюйма.

Например, чтобы сместить строку на один дюйм от левого края, нужно подать команду:

<Esc>, '$', 60, 0

В шестнадцатеричном коде эта команда имеет следующий вид: 1Bh, 24h, 3Ch, 00h.

Печать строки битового образа осуществляется по команде «Печатать графическую строку», которая имеет следующий формат:

<Esc>, '*', m, nL, nH, d1, …, dk

Команда имеет следующие параметры:

- m — код режима печати (см. табл. 7.5);
- nL — младший байт числа колонок в строке;
- nH — старший байт числа колонок в строке;
- d1-dk — байты данных строки изображения.

Число передаваемых байтов данных равно количеству колонок в строке изображения, а разрешение, с которым будет отпечатано изображение, определяется значением параметра m. В сводной табл. 7.5 перечислены коды режимов печати битового образа для принтеров EPSON и совместимых с ними (то есть поддерживающих набор команд ESC/P или ESC/P2) принтеров производства других фирм. Некоторые режимы матричных принтеров имеют в документации Epson индивидуальные названия, список которых приведен в табл. 7.6.

Таблица 7.5. Режимы растровой печати Epson-совместимых матричных принтеров

Код (m)	Горизонтальная плотность	Вертикальная плотность			Печать соседних точек	Точек в колонке	Байтов в колонке
		9 игл	24 иглы	48 игл			
0	60	72	60	60	Да	8	1
1	120	72	60	60	Да	8	1
2	120	72	60	60	Нет	8	1
3	240	72	60	60	Нет	8	1
4	80	72	60	60	Да	8	1
5	72	72	Нет	Нет	Да	8	1
6	90	72	60	60	Да	8	1
7	144	72	Нет	Нет	Да	8	1
32	60	Нет	180	180	Да	24	3
33	120	Нет	180	180	Да	24	3
38	90	Нет	180	180	Да	24	3
39	180	Нет	180	180	Да	24	3
40	360	Нет	180	180	Нет	24	3
64	60	Нет	Нет	360	Да	48	6
65	120	Нет	Нет	360	Да	48	6
70	90	Нет	Нет	360	Да	48	6
71	180	Нет	Нет	360	Да	48	6
72	360	Нет	Нет	360	Нет	48	6
73	360	Нет	Нет	360	Да	48	6

Таблица 7.6. Специфические наименования режимов печати Epson, встречающиеся в документации

Код	Режим печати
0	Обычный 8-игольчатый
1	8-игольчатый с удвоенной плотностью
2	8-игольчатый ускоренный с удвоенной плотностью
3	8-игольчатый с учетверенной плотностью
4	ЭЛТ-I

Код	Режим печати
5	Плоттер
6	ЭЛТ-II
7	Плоттер с удвоенной плотностью
32	24-игольчатый обычный
33	24-игольчатый с удвоенной плотностью
38	ЭЛТ-III
39	24-игольчатый с утроенной плотностью
40	24-игольчатый с шестикратной плотностью

Предельная ширина области печати для принтеров формата А4 равна 8 дюймам, а для принтеров формата А3 — 11 дюймам. Максимально возможное число точек в растровой строке определяется произведением ширины области печати на горизонтальную плотность печати. Например, в режиме 0 на принтере формата А4 предельное число точек в строке равно 480.

ВНИМАНИЕ

В некоторых режимах невозможна печать соседних черных точек, то есть вывод данных осуществляется через одну точку — если две точки изображения идут по горизонтали подряд, друг за другом, то печатается только первая их них. Такие режимы в табл. 7.5 помечены знаком «минус» в колонке «Печать соседних точек».

Плотность печати по вертикали для 9-игольчатых матричных принтеров составляет 72 точки на дюйм, а для 24-игольчатых и струйных — 60, 180 или 360 точек на дюйм. Следовательно, в режимах с одинаковыми кодовыми номерами коэффициенты деформации изображения струйных принтеров отличаются от коэффициентов 9-игольчатых. Например, для 9-игольчатых пропорциональная печать 1:1 возможна в режиме 5, а для струйных — в режимах 0, 39, 73. Поскольку плотность печати 9-игольчатых принтеров не совпадает с плотностью 24-игольчатых и струйных, при использовании старого программного обеспечения для печати на новых принтерах наблюдается искажение масштабов изображения.

Рассмотрим в качестве примера команду для печати в режиме 0 верхнего изображения, показанного на рис. 7.7. В шестнадцатеричном коде команда будет выглядеть следующим образом (курсивом отмечены байты данных):

1Bh, 2Ah, 00h, 12h, 00h, *10h, 1Fh, 21h, 2Dh, 4Dh, C1h, 4Fh, 2Fh, 21h, 1Fh, 10h, 00h, 00h, 38h, 44h, 47h, 44h, 38h*

Здесь передается 18 байт данных, а следовательно nL = 12h, nH = 0.

В конце каждой графической строки должны передаваться команды «возврат каретки» и «перевод строки» (0Dh, 0Ah). После завершения печати изображения нужно подать команду «перевод формата» (0Ch), чтобы извлечь из принтера отпечатанную страницу.

ПРИМЕЧАНИЕ

В режиме битового образа допускается совместная печать графики и текста, причем можно не только печатать вперемежку текстовые и графические строки, но и выводить графику и текст в одной строке. На практике, однако, такие трюки не применяются из-за чрезмерной «заумности» расчетов, нужных для их реализации.

Листинг 7.4 содержит две вспомогательные процедуры, которые мы будем использовать в приводимых ниже тестовых примерах для различных способов печати: процедуру захвата русского шрифта GrabRusFont и процедуру рисования графического изображения шрифта в режиме VGA 320×200 ShowRusFont.

Листинг 7.4. Процедуры для захвата русского шрифта и отображения его в режиме 320×200

```
DATASEG
; Буфер для сохранения шрифта (16x256 байт)
Font8x16 DB 4096 DUP(?)
; Позиция отображаемого символа
FontString DW ? ;номер строки шрифта
FontColumn DW ? ;номер колонки шрифта
ENDS

CODESEG
;****************************************************
;* СЧИТЫВАНИЕ "РУССКОГО" ШРИФТА ИЗ ВИДЕОКОНТРОЛЛЕРА *
;****************************************************
PROC GrabRusFont near
        pushad
; Перепрограммировать синхронизатор
        cli
        mov     DX,3C4h
        ; Установить последовательную адресацию
        ; ячеек видеопамяти
        mov     AX,0704h
        out     DX,AX
        sti
; Перепрограммировать графический контроллер
        mov     DX,3CEh
        ; Выбрать для считывания плоскость 2
        mov     AX,0204h
```

```
        out     DX,AX
        ; Запретить четную-нечетную адресацию
        mov     AX,0005h
        out     DX,AX
        ; Установить окно доступа по адресу A0000h
        mov     AX,0006h
        out     DX,AX

; Скопировать шрифт в буфер Font8x16
        mov     AX,0A000h
        mov     ES,AX
        mov     SI,0
        mov     BX,offset FontBx16
        mov     DX,256
@@M0:   mov     CX,16
@@M1:   mov     AL,[ES:SI]
        mov     [BX],AL
        inc     BX
        inc     SI
        loop    @@M1
        add     SI,16
        dec     DX
        jnz     @@M0
        popad
        ret
ENDP GrabRusFont

;**********************************************
;* ОТОБРАЗИТЬ ШРИФТ НА ЭКРАНЕ В РЕЖИМЕ 320X200 *
;**********************************************
PROC ShowRusFont near
        pusha
        push    ES
        mov     AX,0A000h
        mov     ES,AX
        mov     SI,offset Font8x16
        xor     DI,DI
        mov     [FontString],0
@@m0:   mov     [FontColumn],0
        push    DI
@@m1:   ; Отобразить очередной символ
        mov     AH,16   ;число строк в маске символа
@@m2:   ; Отобразить строку изображения символа
        lodsb           ;загрузить очередной байт маски
        mov     CX,8
@@m3:   ; Вывести на экран очередную точку изображения
        rol     AL,1
        jnc     @@m4
        mov     [byte ptr ES:DI],15
@@m4:   inc     DI
```

продолжение ⇨

Листинг 7.4 (*продолжение*)

```
        loop    @@m3
        add     DI,320-8
        dec     AH
        jnz     @@m2
        sub     DI,320*16-8-2
        inc     [FontColumn]
        cmp     [FontColumn],32
        jb      @@m1
        pop     DI
        add     DI,320*(16+4)
        inc     [FontString]
        cmp     [FontString],8
        jb      @@m0
        pop     ES
        popa
        ret
ENDP ShowRusFont
ENDS
```

В листинге 7.5 приведен тест на совместимость принтера с набором команд Epson ESC/P2. Программа Test_Matrix_On_LPT1 осуществляет печать графического изображения набора символов шрифта 8×16 в режиме битового образа. Программа использует универсальные процедуры ввода-вывода из главы 1 «Работа с клавиатурой», процедуры захвата шрифта и рисования его изображения в режиме VGA 320×200 из листинга 7.4, а также процедуры вывода команд и данных на принтер из листинга 7.1.

Листинг 7.5. Тест для матричных и струйных принтеров на совместимость с режимом печати битового образа EPSON

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Номер печатаемой строки изображения
PrintingString DW ?
; Номер печатаемого байта
PrintingByte DW ?
; Текстовые сообщения
```

```
Txt1 DB LIGHTCYAN,0,19
     DB "ПЕЧАТЬ КОДОВОЙ ТАБЛИЦЫ РУССКОГО ШРИФТА DOS",0
     DB LIGHTCYAN,2,20
     DB "НА МАТРИЧНОМ EPSON-СОВМЕСТИМОМ ПРИНТЕРЕ",0
     DB LIGHTGREEN,12,11
     DB "Включите принтер, вставьте "
     DB "бумагу, установите режим ON-LINE",0
     DB LIGHTGREEN,14,17
     DB "(будет произведен переход в графический режим)",0
     DB YELLOW,24,14,"Нажмите любую клавишу и "
     DB "ждите завершения печати",0
Txt2 DB LIGHTGREEN,12,28,"Печать шрифта завершена",0
     DB YELLOW,24,29,"Нажмите любую клавишу",0

; КОМАНДЫ ДЛЯ ПРИНТЕРА
; Установить межстрочное расстояние 8 точек
SetLineSpacing   DB 3, 1Bh,'A',8
; Вывести 320 точек по горизонтали (256+64)
SendBitImageData DB 5, 1Bh,'*',1,64,1
ENDS

SEGMENT sseg para stack 'STACK'
       DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC Test_Matrix_On_LPT1
       mov     AX,DGROUP
       mov     DS,AX
       mov     [CS:MainDataSeg],AX
; Считать шрифт из видеопамяти
       call    GrabRusFont
; Установить текстовый режим и очистить экран
       mov     AX,3
       int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
       mov     [ScreenString],25
       mov     [ScreenColumn],0
       call    SetCursorPosition
; Вывести текстовые сообщения на экран
       MShowColorText 5,Txt1
       call    GetChar

; Установить видеорежим VGA 320×200, 256 цветов
       mov     AX,13h
       int     10h
; Отобразить шрифт
```

продолжение ↗

Листинг 7.5 (*продолжение*)

```
        call    ShowRusFont
; Установить межстрочное расстояние 8 точек
        mov     SI,offset SetLineSpacing
        call    OutCommandToLPT1
; Настроить пару регистров ES:SI на видеопамять
        mov     AX,0A000h
        mov     ES,AX
        xor     SI,SI  ;обнулить SI
; Сбросить счетчик строк
        mov     [PrintingString],0

; ОСНОВНОЙ ЦИКЛ (ПО ПЕЧАТАЕМЫМ СТРОКАМ)
; Печать осуществляется в инверсной форме (светлые точки
; экрана при печати отображаются черными и наоборот).
; Вывод изображения на матричный принтер выполняется
; слева направо, сверху вниз, строками шириной по
; восемь точек.
@@P0:
; Запомнить начало очередной строки в видеопамяти
        push    SI
; Включить графический режим печати
        push    SI
        mov     SI,offset SendBitImageData
        call    OutCommandToLPT1
        pop     SI
; Сбросить счетчик байтов
        mov     [PrintingByte],0
; Цикл по печатаемым байтам
@@P1:   push    SI
        mov     CX,8   ;счетчик точек в байте
        xor     AL,AL  ;обнулить байт
; Цикл по печатаемым точкам
@@P2:   shl     AL,1   ;сдвинуть разряды влево
        cmp     [byte ptr ES:SI],0 ;цвет точки?
        je      @@P3   ;пропустить черную точку
        or      AL,1   ;"поставить" точку
@@P3:   ; Перейти на следующую строку изображения
        add     SI,320
        loop    @@P2
        ; Вывести байт на принтер
        call    OutCharToLPT1
        ; Перейти к следующему байту
        pop     SI
        inc     SI
        inc     [PrintingByte]
        cmp     [PrintingByte],320
        jl      @@P1
; Перейти на следующую строку
        ; Послать на принтер коды возврата
```

```
        ; каретки и перевода строки
        mov     AL,0Dh ;возврат каретки
        call    OutCharToLPT1
        mov     AL,0Ah ;перевод строки
        call    OutCharToLPT1
        ; Вычислить начало следующей группы из
        ; восьми строк в видеопамяти
        pop     SI
        add     SI,320*8
        ; Увеличить счетчик отпечатанных строк
        inc     [PrintingString]
        cmp     [PrintingString],160/8
        jl      @@P0
; Послать на принтер коды завершения страницы
        mov     AL,0Ch ;перевод формата
        call    OutCharToLPT1
; Переустановить текстовый режим
        mov     ax,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        call    SetCursorPosition
; Вывести сообщение о завершении печати
        MShowColorString Txt1
        MShowColorText 2,Txt2
        call    GetChar
@@End:  ; Переустановить текстовый режим
        mov     ax,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP Test_Matrix_On_LPT1
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуры вывода символа и посылки
; команды на принтер
include "list7_01.inc"
; Подключить процедуры для захвата русского шрифта и
; отображения его в режиме 320x200
include "list7_04.inc"

END
```

ПРИМЕЧАНИЕ

Для запуска теста пригоден любой AT-совместимый компьютер. Принтер с Epson-совместимым набором команд должен быть подключен к порту LPT1.

СОВЕТ

Прежде чем запустить тест на принтере с автоматической подачей бумаги, извлеките бумагу из лотка и оставьте в нем только один лист (если принтер не поддерживает набор команд EPSON, то он воспримет передаваемую информацию как бессмысленный набор ASCII-символов и попытается их распечатать в текстовом режиме, что может привести к бесполезному расходованию бумаги).

Набор команд Epson для печати в растровом режиме

В отличие от группы команд ESC/P, ставших международным стандартом, набор команд для печати в растровом режиме используется только принтерами Epson. Общая тенденция компьютерной отрасли к полному отказу от текстового режима и переходу на растровую печать коснулась и всех младших моделей струйных принтеров Epson: они не поддерживают текстовый режим и печать битовых образов.

Прежде всего следует отметить, что в растровом режиме доступны для использования некоторые команды общего назначения из набора ESC/P:

- инициализация,
- установка абсолютной позиции по горизонтали,
- перевод строки,
- перевод формата,
- возврат каретки.

Команда перевода строки <LF> перемещает курсор по оси X к левому краю поля печати, а значение координаты по оси Y увеличивает на заданную величину разделительного интервала. Если координата по оси Y выходит за пределы поля печати, то текущая страница извлекается из принтера, загружается следующая страница, а значение координаты по Y устанавливается в соответствии с верхней границей поля печати.

По команде перевода формата <FF> распечатывается содержимое буфера печати, буфер освобождается, после чего производится извлечение текущей страницы и загрузка новой. Для новой страницы координаты X и Y устанавливаются в соответствии с заданным положением левого верхнего угла области печати.

Команда возврата каретки <CR> перемещает курсор к левому краю поля печати.

Набор растровых команд Epson [45, 48–50] развивается по мере совершенствования струйных принтеров и появления новых технологий печати, но в этом наборе имеется некоторое подмножество универсальных команд, которые поддерживаются всеми или почти всеми находящимися в данный момент в эксплуатации моделями струйных принтеров Epson. Ниже мы будем рассматривать только универсальные команды, которые перечислены в табл. 7.7. Параметры команд, передаваемые в двоичном коде, будем обозначать прописными латинскими буквами (если параметр является 16-разрядным числом, то буквой L помечен младший байт, буквой H — старший байт).

Таблица 7.7. Команды растрового режима печати Epson

Назввние команды	Esc-последоввтвльность
Переключить принтер в графический режим	<ESC> '(' 'G' 1 0 1
Установить размер единицы перемещения	<ESC> '(' 'U' 1 0 m
Включить/выключить однонаправленный режим печати	<ESC> 'U' n
Выбрать монохромный/цветной режим печати	<ESC> '(' 'K' 1 0 0 n
Выбрать размер точки	<ESC> '(' 'e' 2 0 0 d
Задать длину листа	<ESC> '(' 'C' 2 0 mL mH
Задать формат страницы	<ESC> '(' 'c' 4 0 tL tH bL bH
Установить абсолютную позицию печати по вертикали	<ESC> '(' 'V' 2 0 mL mH
Выбрать цвет печати	<ESC> 'r' n
Установить новую позицию печати по горизонтали относительно текущей	<ESC> '\' nL nH
Печать растровой графики	<ESC> '.' c v h m nL nH d1...dk
Установить новую позицию печати по вертикали относительно текущей	<ESC> '(' 'v' 2 0 mL mH

В руководствах для программистов фирма Epson рекомендует придерживаться определенного порядка выполнения операция при использовании растрового режима печати. В соответствии с рекомендациями Epson процесс печати делится на следующие этапы:

- этап 1 — инициализация,
- этап 2 — выбор способа печати,
- этап 3 — выбор формата страницы;
- этап 4 — передача изображения;

- этап 5 — переход на новую страницу;
- этап 6 — завершение печати.

Этапы 1 и 2 выполняются однократно: в результате выполнения этих этапов задаются параметры печати, относящиеся ко всему документу.

Этап инициализации предполагает выполнение группы из трех команд в следующем порядке:

1. команда инициализации;
2. команда переключения в графический режим;
3. команда установки размера единицы перемещения.

Команда инициализации, как уже было указано выше, имеет следующий формат:

```
<Esc>, '@'
```

Команда переключения в графический режим переводит принтер в режим обработки графических команд. Она имеет следующий формат:

```
<Esc>, '(', 'G', 1, 0, 1
```

Команда установки размера единицы перемещения задает величину шага перемещения печатающей головки:

```
<Esc>, '(', 'U', 1, 0, m
```

После выполнения команды величина шага будет составлять m/3600 дюйма. Параметр m может принимать одно из следующих значений: 5, 10, 20, 30, 40, 50, 60.

ВНИМАНИЕ

После инициализации принтера величина шага перемещения в растровом режиме по умолчанию составляет 1/360 дюйма.

Команда установки размера единицы перемещения не является обязательной, и ее можно опустить, если заданное по умолчанию значение шага перемещения подходит для используемого режима печати.

Этап выбора способа печати предполагает выполнение группы из трех команд в следующем порядке:

1. команда включения и отключения однонаправленного режима;
2. команда выбора режима печати;
3. команда выбора размера точки.

Команда включения и отключения однонаправленного режима позволяет программисту выбирать между качественной и черновой

печатью. Однонаправленный режим позволяет получить более качественное изображение при печати фотографий, чертежей и рисунков за счет снижения (вдвое) скорости печати. Двунаправленный режим применяется для быстрой черновой печати; кроме того, если высота шрифта меньше высоты линейки сопел печатающей головки, в двунаправленном режиме можно добиться качественной печати текста.

Команда включения и отключения однонаправленного режима печати имеет следующий формат:

<Esc>, 'U', n

Значения параметра n:

- 0 — установить двунаправленный режим печати;
- 1 — установить однонаправленный режим печати.

Если используется черновой (двунаправленный) режим печати, команду включения и отключения однонаправленного режима можно опустить.

Команда выбора режима печати позволяет программисту выбрать либо монохромный, либо цветной режим печати. Эта команда имеет следующий формат:

<Esc>, '(', 'K', 1, 0, 0, n

Параметр n задает режим печати:

- 0 — режим, принятый по умолчанию (для цветных принтеров — цветной);
- 1 — монохромный (черно-белый) режим;
- 2 — цветной режим.

Команда выбора размера точки позволяет выбрать размер точки изображения (размер чернильной капли). Формат команды следующий:

<Esc>, '(', 'e', 2, 0, 0, d

Допустимые значения параметра d различны для разных моделей принтеров: общим является только значение 0, соответствующее размеру точки, принятому по умолчанию. Программист может либо опустить данную команду, либо найти нужное значение d в документации на конкретную группу моделей принтеров.

Этап выбора формата страницы предполагает выполнение группы из двух команд в следующем порядке:

1) команда установки длины листа;
2) команда установки формата страницы.

Команда установки длины листа задает длину листа бумаги в единицах перемещения. Формат этой команды следующий:

```
<Esc>, '(', 'C', 2, 0, mL, mH
```

Через mL и mH в данном случае обозначены младший и старший байты слова, задающего длину листа в единицах перемещения. По умолчанию задана длина листа 22 дюйма.

Команда установки формата страницы задает расстояние (в единицах перемещения) верхней и нижней границ области печати от верхней кромки листа. Команда имеет следующий формат:

```
<Esc>, '(', 'c', 4, 0, tL, tH, bL, bH
```

Через tL и tH обозначены младший и старший байты слова, задающего расстояние до верхней границы области печати, через bL и bH — младший и старший байты слова, задающего расстояние до нижней границы. По умолчанию верхняя граница расположена на расстоянии 1/3 дюйма от верхней кромки листа, а нижняя — на расстоянии 22 дюйма (совпадает с нижней кромкой).

При использовании бумаги формата A4 этап выбора формата страницы можно опустить.

Этап передачи изображения начинается с установки положения изображения относительно верхней границы области печати. Эта операция может выполняться при помощи команды установки абсолютного положения по вертикали.

Команда установки абсолютного положения по вертикали определяет положение начальной строки изображения относительно верхней границы области печати. Эта команда имеет следующий формат:

```
<Esc>, '(', 'V', 2, 0, mL, mH
```

Параметр m в данной команде задает абсолютную координату по Y в единицах перемещения.

Собственно передача изображения заключается в последовательной передаче всех цветовых компонентов для каждой строки изображения. Следовательно, прежде чем передать данные одного цветового компонента, необходимо задать цвет этого компонента (код цветовой плоскости, в которую ведется запись передаваемых данных). Выбор цвета компонента осуществляется с помощью команды, имеющей следующий формат:

```
<Esc>, 'r', n
```

Параметр n в данной команде содержит код цвета, который может принимать одно из следующих значений:

- 0 — черный,
- 1 — пурпурный,
- 2 — бирюзовый,
- 4 — желтый.

По умолчанию (после инициализации) значение n равно нулю, что соответствует черному цвету. Следовательно, при выполнении печати монохромного черно-белого изображения команду установки цвета можно не использовать.

Вторая операция — установка начального положения курсора по горизонтали при помощи какой-либо из команд позиционирования по оси X. В эту группу входят следующие команды:

- команда установки абсолютной позиции по горизонтали;
- команда относительного перемещения по горизонтали;
- команда возврата каретки.

Команда установки абсолютной позиции по горизонтали, как было указано выше, имеет следующий формат:

<Esc>, '$', nL, nH

Параметр n в данной команде задает абсолютную координату по X в единицах перемещения.

Команда относительного перемещения по горизонтали смещает положение области печати на заданное значение единиц перемещения от текущей точки. Команда имеет следующий формат:

<Esc>, '\', nL, nH

Параметр n в данной команде представляет собой 16-разрядное двоичное число со знаком, которое задает смещение по оси X относительно текущей позиции печати (в единицах перемещения).

Команда возврата каретки перемещает позицию печати в начало строки (к левой границе области печати).

Передача растровых данных для одного цветового компонента строки или группы строк изображения выполняется по команде печати растровой графики, которая имеет следующий формат:

<Esc>, '.', c ,v, h, m, nL, nH, $d_1...d_k$

Команда передачи растровых данных имеет параметры:

- c — признак сжатия графических данных (0 — сжатие не используется, 1 — используется сжатие по алгоритму RLE);
- v — код разрешения по вертикали (задает разрешение v/3600 точек/дюйм);
- h — код разрешения по горизонтали (задает разрешение h/3600 точек/дюйм);
- m — количество передаваемых растровых линий;
- n — количество точек в одной строке изображения;
- d_1, ..., d_k — байты данных.

С помощью этой команды можно передавать не только одну строку, но и группу из m растровых строк для одной цветовой плоскости. Структура такой полосы, состоящей из нескольких строк, показана на рис. 7.9.

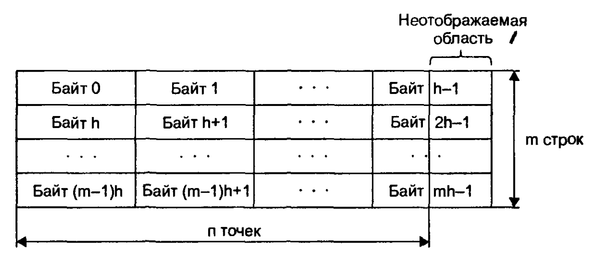


Рис. 7.9. Структура полосы изображения, которую выводит команда передачи растровых данных

Для разных принтеров являются допустимыми различные комбинации параметров v, h и m. Большинство моделей для параметра m допускает использование значений 1, 8 и 24 (значение 1 является предпочтительным), а для величин v и h — комбинации v = 10, h = 10 (разрешение 360×360).

Исключением из этого правила является принтеры C20X–C40X, которые могут печатать только с разрешением 120×360 точек/дюйм. В монохромном режиме они допускает комбинацию параметров v = 30, h = 10, m = 48 (за одну операцию передается 48 линий растра), а в цветном режиме — комбинацию v = 30, h = 10, m = 15 (за одну операцию передается 15 строк цветовой плоскости).

Количество байтов в одной строке изображения равняется округленному вверх (до ближайшего целого) частному от деления количества точек в строке на 8. Соответственно, общее количество передаваемых командой байтов данных вычисляется по формуле

```
k = m×int((n + 7) / 8).
```

Переход на следующую строку растра осуществляется при помощи команды относительного перемещения по вертикали, которая имеет следующий формат:

```
<Esc>, '(', 'v', 2, 0, mL, mH
```

Параметр m в данной команде представляет собой 16-разрядное двоичное число со знаком, которое задает смещение по оси Y относительно текущей позиции печати (в единицах перемещения).

Команда относительного перемещения по вертикали применяется также при переходе к следующему по порядку изображению, если на одной странице имеется несколько изображений.

Таким образом, передача одной строки изображения представляет собой цикл по цветовым компонентам, внутри которого выполняется следующая последовательность команд:

1. установить код цвета для передаваемой компоненты;
2. переместить курсор по горизонтали в начало растровой строки;
3. передать данные цветовой компоненты.

Процесс передачи изображения заключается в последовательном выполнении двух операций для каждой передаваемой строки:

1. передать все цветовые компоненты строки;
2. перейти на следующую строку при помощи команды относительного перемещения по вертикали.

Если передача растровых данных ведется построчно, для перехода на следующую строку растра нужно использовать относительное смещение (положительное) на одну единицу перемещения. Если передача растровых данных ведется полосами из нескольких строк, для перехода на следующую строку растра нужно использовать смещение на m единиц, где m — количество растровых строк в полосе.

В листинге 7.6 приведена программа EpsonStylus_BW, демонстрирующая печать черно-белого изображения набора символов шрифта 8×16 в растровом режиме. Программа использует универсальные процедуры ввода-вывода из главы 1 «Работа с клавиатурой», процедуры захвата шрифта и рисования его изображения в режиме

VGA 320×200 из листинга 7.4, а также процедуры вывода команд и данных на принтер из листинга 7.1.

Листинг 7.6. Тест для струйных принтеров EPSON Stylus: печать в черно-белом растровом режиме

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Номер печатаемой строки изображения
PrintingString DW ?
; Номер печатаемого байта
PrintingByte DW ?
; Текстовые сообщения
Txt1 DB LIGHTCYAN,0,19
     DB "ПЕЧАТЬ КОДОВОЙ ТАБЛИЦЫ РУССКОГО ШРИФТА DOS",0
     DB LIGHTCYAN,2,23
     DB "НА СТРУЙНОМ ПРИНТЕРЕ EPSON STYLUS",0
     DB LIGHTGREEN,12,11
     DB "Включите принтер, вставьте "
     DB "бумагу, установите режим ON-LINE",0
     DB LIGHTGREEN,14,17
     DB "(будет произведен переход в графический режим)",0
     DB YELLOW,24,14,"Нажмите любую клавишу и "
     DB "ждите завершения печати",0
Txt2 DB LIGHTGREEN,12,28,"Печать шрифта завершена",0
     DB YELLOW,24,29,"Нажмите любую клавишу",0

; КОМАНДЫ ДЛЯ ПРИНТЕРА
; Инициализировать принтер
PrnInitialization   DB 2, 1Bh,'@'
; Установить графический режим
SelectGraphicsMode  DB 6, 1Bh,'(','G',1,0,1
; Выбор монохромного режима
MonochromeSelection DB 7, 1Bh,'(','K',1,0,0,1
; Выбор разрешения 360x360
```

```
SetResolution      DB 9, 1Bh,'(','D',4,0,14,1,1,1
; Печать растровой графики (320 точек в строке)
PrintRasterData    DB 8, 1Bh,'.',0,10,10,1,64,1
; Перевод строки
SetRelVertPosition DB 7, 1Bh,'(','v',2,0,1,0
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC EpsonStylus_BW
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Считать шрифт из видеопамяти
        call    GrabRusFont
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 5,Txt1
        call    GetChar
; Установить видеорежим VGA 320×200, 256 цветов
        mov     AX,13h
        int     10h
; Отобразить шрифт
        call    ShowRusFont
; Инициализировать принтер
        mov     SI,offset PrnInitialization
        call    OutCommandToLPT1
; Включить графический режим печати
        mov     SI,offset SelectGraphicsMode
        call    OutCommandToLPT1
; Выбрать монохромный режим
        mov     SI,offset MonochromeSelection
        call    OutCommandToLPT1
; Выбрать разрешение 360×360
        mov     SI,offset SetResolution
        call    OutCommandToLPT1
; Настроить пару регистров ES:DI на видеопамять
        mov     AX,0A000h
        mov     ES,AX
        xor     DI,DI   ;обнулить DI
; Сбросить счетчик строк растра
```

продолжение ⇒

Листинг 7.6 (*продолжение*)

```
        mov     [PrintingString],0

; ОСНОВНОЙ ЦИКЛ (ПО ПЕЧАТАЕМЫМ СТРОКАМ)
; Печать осуществляется в инверсной форме (светлые точки
; экрана при печати отображаются черными и наоборот).
; Вывод изображения на принтер выполняется по строкам
; растра, слева направо, сверху вниз.
@@P0:
; Задать длину строки 40 байт (320/8)
        mov     SI,offset PrintRasterData
        call    OutCommandToLPT1
; Сбросить счетчик байтов
        mov     [PrintingByte],0
; Цикл по печатаемым байтам
@@P1:   mov     CX,8    ;счетчик точек в байте
        xor     AL,AL   ;обнулить байт
; Цикл по печатаемым точкам
@@P2:   shl     AL,1    ;сдвинуть разряды влево
        cmp     [byte ptr ES:DI],0 ;цвет точки?
        je      @@P3    ;пропустить черную точку
        or      AL,1    ;"поставить" точку
@@P3:   ; Перейти на следующую точку
        inc     DI
        loop    @@P2
        ; Вывести байт на принтер
        call    OutCharToLPT1
        ; Перейти к следующему байту
        inc     [PrintingByte]
        cmp     [PrintingByte],40
        jl      @@P1
; Перейти на следующую строку растра принтера
        mov     SI,offset SetRelVertPosition
        call    OutCommandToLPT1
; Послать на принтер команду возврата каретки
        mov     AL,0Dh
        call    OutCharToLPT1
; Перейти на следующую строку экранного изображения
        inc     [PrintingString]
        cmp     [PrintingString],160
        jl      @@P0

; Послать на принтер коды завершения страницы
        mov     AL,0Ch ;перевод формата
        call    OutCharToLPT1
; Инициализировать принтер
        mov     SI,offset PrnInitialization
        call    OutCommandToLPT1

; Переустановить текстовый режим
        mov     ax,3
```

```
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        call    SetCursorPosition
; Вывести сообщение о завершении печати
        MShowColorString Txt1
        MShowColorText 2,Txt2
        call    GetChar
@@End:  ; Переустановить текстовый режим
        mov     ax,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP EpsonStylus_BW
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуры вывода символа и посылки
; команды на принтер
include "list7_01.inc"
; Подключить процедуры для захвата русского шрифта и
; отображения его в режиие 320x200
include "list7_04.inc"

END
```

После распечатки результатов работы тестового примера из листинга 7.6 вы увидите на листе бумаги множество символов очень маленького размера: шрифт 8×16 является слишком мелким для печати с разрешением 360×360 точек/дюйм. Для качественной печати текста при таком высоком разрешении размер матрицы символа должен быть в несколько раз больше, а для черновой печати можно обойтись простым масштабированием шрифта.

Листинг 7.7 содержит процедуру CreateRasterImage, формирующую растровое изображение текстовой строки. Процедура использует шрифт 8×16, но растягивает символы в четыре раза только по горизонтали: масштабирование по вертикали будет выполняться в программе печати текста путем четырехкратного повторения каждой строки при выводе текста на печать.

Листинг 7.7. Подпрограмма, создающая монохромное растровое изображение заданной текстовой строки

```
DATASEG
; Буфер для сохранения монохромного растрового
; изображения строки текста
RowImage DB 4096 DUP(?) ;256 байт, 16 строк
```

продолжение ⇨

Листинг 7.7 (*продолжение*)

```
ENDS

CODESEG
;*****************************************************
;* СФОРМИРОВАТЬ РАСТРОВОЕ ИЗОБРАЖЕНИЕ ТЕКСТОВОЙ СТРОКИ *
;* Передаваемые параметры:                           *
;* DS:SI - указатель на строку, ограниченную нулем.  *
;*****************************************************
PROC CreateRasterImage near
        pusha
        push    ES
        cld
; Очистить буфер строки
        mov     AX,DS
        mov     ES,AX
        mov     DI,offset RowImage
        mov     CX,4096/2
        xor     AX,AX
        rep     stosw

; ЦИКЛ ВЫВОДА СИМВОЛОВ
        mov     DI,offset RowImage
        mov     CX,64 ;ограничитель длины строки
@@NextChar:
; Загрузить код символа
        lodsb
        and     AL,AL ;конец строки?
        jz      @@Exit
        push    DI
        push    SI
; Вычислить начальную позицию маски символа
        mov     SI,offset Font8x16
        xor     AH,AH
        shl     AX,4  ;смещение символа от начала шрифта
        add     SI,AX
; Цикл по строкам растра
        mov     DH,16
@@NextMaskByte:
        lodsb
; Цикл по точкам растра
        mov     DL,4
@@NextBite:
; По горизонтали растягиваем символы в 4 раза
        ; Нечетные биты соответствуют старшей тетраде
        test    AL,80h
        jz      @@NoD1
        or      [byte ptr ES:DI],0F0h
        ; Четные биты соответствуют младшей тетраде
@@NoD1: test    AL,40h
        jz      @@NoD2
```

```
        or      [byte ptr ES:DI],0Fh
@@NoD2: rol     AL,2    ;проверить следующую пару битов
        inc     DI
        dec     DL
        jnz     @@NextBite
        add     DI,256-4 ;перейти на следующую строку
        dec     DH
        jnz     @@NextMaskByte
        pop     SI
        pop     DI
        add     DI,4
        loop    @@NextChar
@@Exit: pop     ES
        popa
        ret
ENDP CreateRasterImage
```

Листинг 7.8 содержит программу PrintColorText, которая выполняет печать группы текстовых строк различного цвета. Для формирования растрового изображения текстовой строки программа использует процедуру CreateRasterImage из листинга 7.7. Вывод одного цветового компонента строки выполняется процедурой Out_Color_Component, которая вызывается из процедуры печати изображения текстовой строки Out_Text_String. Процедура Out_Text_String обращается к процедуре Out_Color_Component для передачи одной и той же группы данных в каждую из используемых (для формирования заданного цвета строки) цветовых плоскостей, а неиспользуемые плоскости пропускаются. Каждую строку изображения, созданного процедурой CreateRasterImage, процедура Out_Text_String дублирует по вертикали четыре раза.

Листинг 7.8. Тест для струйных принтеров EPSON Stylus: печать цветного текста в растровом режиме

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
```


продолжение ⇨

Листинг 7.8 (*продолжение*)

```
ENDS

DATASEG
; Текстовые сообщения
Txt1 DB LIGHTCYAN,0,19
     DB "ПЕЧАТЬ ЦВЕТНОГО ТЕКСТА В РАСТРОВОМ РЕЖИМЕ",0
     DB LIGHTCYAN,2,23
     DB "НА СТРУЙНОМ ПРИНТЕРЕ EPSON STYLUS",0
     DB LIGHTGREEN,12,11
     DB "Включите принтер, вставьте "
     DB "бумагу, установите режим ON-LINE",0
     DB YELLOW,24,14,"Нажмите любую клавишу и "
     DB "ждите завершения печати",0
; КОМАНДЫ ДЛЯ ПРИНТЕРА
; Инициализировать принтер
PrnInitialization  DB 2, 1Bh,'@'
; Установить графический режим
SelectGraphicsMode DB 6, 1Bh,'(','G',1,0,1
; Выбор цветного режима
ColorSelection     DB 7, 1Bh,'(','K',1,0,0,0
; Выбор разрешения 360x360
SetResolution      DB 9, 1Bh,'(','D',4,0,14,1,1,1
; Выбор цвета для печати
SelectBlack        DB 3, 1Bh,'r',0
SelectMagenta      DB 3, 1Bh,'r',1
SelectCyan         DB 3, 1Bh,'r',2
SelectYellow       DB 3, 1Bh,'r',4
; Печать растровой графики (2048 точек в строке)
PrintRasterData    DB 8, 1Bh,'.',0,10,10,1,0,8
; Перевод строки
SetRelVertPosition DB 7, 1Bh,'(','v',2,0,1,0
; Текст для печати на принтере
CyanRow    DB "Голубая строка",0
MagentaRow DB "Пурпурная строка",0
YellowRow  DB "Желтая строка",0
RedRow     DB "Красная строка",0
GreenRow   DB "Зеленая строка",0
BlueRow    DB "Синяя строка",0
BlackRow   DB "Черная строка",0
; Код цвета строки в формате CMY
; (бит 0 - Magenta, 1 - Cyan, бит 2 - Yellow)
TextStringColor DB ?
ENDS

CODESEG
;****************************
;* Основной модуль програмы *
;****************************
```

```
PROC PrintColorText
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Считать шрифт из видеопамяти
        call    GrabRusFont
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 4,Txt1
        call    GetChar
; Инициализировать принтер
        mov     SI,offset PrnInitialization
        call    OutCommandToLPT1
; Включить графический режим печати
        mov     SI,offset SelectGraphicsMode
        call    OutCommandToLPT1
; Выбрать цветной режим
        mov     SI,offset ColorSelection
        call    OutCommandToLPT1
; Выбрать разрешение 360x360
        mov     SI,offset SetResolution
        call    OutCommandToLPT1

; Вывести пурпурную строку на принтер
        mov     [TextStringColor],001b
        mov     SI,offset MagentaRow
        call    Out_Text_String
; Вывести голубую строку
        mov     [TextStringColor],010b
        mov     SI,offset CyanRow
        call    Out_Text_String
; Вывести желтую строку на принтер
        mov     [TextStringColor],100b
        mov     SI,offset YellowRow
        call    Out_Text_String
; Вывести красную строку на принтер
        mov     [TextStringColor],101b
        mov     SI,offset RedRow
        call    Out_Text_String
; Вывести зеленую строку на принтер
        mov     [TextStringColor],110b
        mov     SI,offset GreenRow
        call    Out_Text_String
; Вывести синюю строку на принтер
```

продолжение ⇗

Листинг 7.8 (*продолжение*)

```
        mov     [TextStringColor],011b
        mov     SI,offset BlueRow
        call    Out_Text_String
; Вывести черную строку
        mov     [TextStringColor],111b
        mov     SI,offset BlackRow
        call    Out_Text_String

; Послать на принтер коды завершения страницы
        mov     AL,0Ch ;перевод формата
        call    OutCharToLPT1
; Инициализировать принтер
        mov     SI,offset PrnInitialization
        call    OutCommandToLPT1
; Переустановить текстовый режим
        mov     ax,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP PrintColorText

;****************************************************
;*        ОТПЕЧАТАТЬ ЦВЕТНУЮ ТЕКСТОВУЮ СТРОКУ       *
;* Передаваемые параметры:                          *
;* TextStringColor - код цвета строки в формате CMY; *
;* DS:SI - указатель на строку, ограниченную нулем.  *
;****************************************************
PROC Out_Text_String near
        pusha
; Сформировать строку
        call    CreateRasterImage
; ОСНОВНОЙ ЦИКЛ (ПО СТРОКАМ ИЗОБРАЖЕНИЯ)
        mov     BX,0
@@NextRasterRow:
; Продублировать строку четыре раза
        mov     DL,4
@@DuplicateString:
        test    [TextStringColor],1b
        jz      @@NoMagenta
        mov     SI,offset SelectMagenta
        call    OutCommandToLPT1
        call    Out_Color_Component
@@NoMagenta:
        test    [TextStringColor],10b
        jz      @@NoCyan
        mov     SI,offset SelectCyan
        call    OutCommandToLPT1
        call    Out_Color_Component
```

```
@@NoCyan:
        test    [TextStringColor],100b
        jz      @@NoYellow
        mov     SI,offset SelectYellow
        call    OutCommandToLPT1
        call    Out_Color_Component
@@NoYellow:
        ; Перейти на следующую строку растра принтера
        mov     SI,offset SetRelVertPosition
        call    OutCommandToLPT1
        dec     DL
        jnz     @@DuplicateString
; Перейти на следующую строку изображения
        inc     BX
        cmp     BX,16
        jb      @@NextRasterRow
        popa
        ret
ENDP Out_Text_String

;***************************************************
;* ВЫВЕСТИ ОДНУ ЦВЕТОВУЮ КОМПОНЕНТУ СТРОКИ РАСТРА *
;* Передаваемые параметры:                         *
;* BX - номер печатаемой сторки изображения.       *
;***************************************************
PROC Out_Color_Component near
        pusha
; Задать длину строки
        mov     SI,offset PrintRasterData
        call    OutCommandToLPT1
; Настроить SI на печатаемую строку
        mov     SI,offset RowImage
        rol     BX,8
        add     SI,BX
; Вывести строку
        mov     CX,256
@@PrintNextByte:
        lodsb
        call    OutCharToLPT1
        loop    @@PrintNextByte
; Послать на принтер команду возврата каретки
        mov     AL,0Dh
        call    OutCharToLPT1
        popa
        ret
ENDP Out_Color_Component
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
```

продолжение ⇨

Листинг 7.8 (*продолжение*)

```
; Подключить процедуры вывода символа и посылки
; команды на принтер
include "list7_01.inc"
; Подключить процедуры для захвата русского шрифта и
; отображения его в режиме 320x200
include "list7_04.inc"
; Подключить процедуру, формирующую монохромное
; растровое изображение заданной текстовой строки
include "list7_07.inc"

END
```

Командный язык PCL фирмы Hewlett-Packard

Печать в растровом режиме на принтерах Hewlett-Packard может осуществляться при помощи языка PCL [54, 83, 84, 87, 89]. Язык PCL имеет богатый набор команд, однако при растровой печати реально необходимой является лишь небольшая подгруппа из этого набора, приведенная в табл. 7.8. Символ # в командной Esc-последовательности означает числовую константу, записанную в коде ASCII.

Таблица 7.8. Основные команды языка PCL, применяемые при растровой печати

Команда	Параметры команды	Esc-последовательность в ASCII-кодах
Сброс принтера	—	<Esc>E
Установить координату курсора в единицах PCL по оси X	—	<Esc>*p#X
Установить координату курсора в единицах PCL по оси Y	—	<Esc>*p#Y
Включить растровый режим и установить разрешение	75 тчк/дюйм	<Esc>*t75R
	100 тчк/дюйм	<Esc>*t100R
	150 тчк/дюйм	<Esc>*t150R
	300 тчк/дюйм	<Esc>*t300R
	600 тчк/дюйм	<Esc>*t600R
Задать число цветовых плоскостей в строке изображения	Одна плоскость, черно-белый режим	<Esc>*r1U

Команда	Параметры команды	Esc-последовательность в ASCII-кодах
	Три плоскости, цветной режим RGB	<Esc>*r3U
	Три плоскости, цветной режим CMY	<Esc>*r-3U
	Четыре плоскости, цветной режим KCMY	<Esc>*r-4U
Задать ширину изображения в пикселах	—	<Esc>*r#S
Начать вывод растрового изображения	С левого края поля печати текущей строки	<Esc>*r0A
	С текущей позиции курсора в текущей строке	<Esc>*r1A
Установить компрессию растра	Без компрессии	<Esc>*b0M
	Последовательное кодирование	<Esc>*b1M
	TIFF-кодирование	<Esc>*b2M
	Дельта-кодирование	<Esc>*b3M
Передать строку цветоаой плоскости	—	<Esc>*b#V [data]
Передать строку растра	—	<Esc>*b#W [data]
Выход из режима растровой печати	—	<Esc>*rC

Прежде чем начать работу в растровом режиме, необходимо выполнить сброс принтера, чтобы перевести все его параметры в состояние, принятое по умолчанию: вообще говоря, при запуске программы неизвестно, какие изменения в параметры работы принтера внесли ранее выполнявшиеся программы. Команда инициализации имеет следующий вид:

```
<Esc>, 'E'
```

После инициализации принтер находится в текстовом режиме, но команда сброса воздействует и на параметры графического режима: по умолчанию задается черно-белый режим печати с разрешением 75 точек/дюйм.

Устанавливаемые по умолчанию границы области печати располагаются в нескольких миллиметрах от края листа бумаги, поэтому после выполнения сброса обычно требуется сместить начальную точку (левый верхний угол) печатаемого изображения на некоторое расстояние от границ рабочей области. Операцию установки

начального положения курсора нужно выполнить до входа в графический режим: команда позиционирования по оси X в графическом режиме игнорируется. Например, чтобы сместить начальное положение курсора относительно левого верхнего угла рабочей области страницы на два дюйма по горизонтали и один дюйм по вертикали, нужно последовательно подать команды позиционирования по X и по Y:

```
<Esc>, '*', 'p', '6', '0', '0', 'X'
<Esc>, '*', 'p', '3', '0', '0', 'Y'
```

В приведенном примере задаются абсолютные координаты (координаты относительно левого верхнего угла области печати). Если перед числом стоит плюс или минус, то задается смещение относительно текущей позиции курсора.

ПРИМЕЧАНИЕ

В командах, предназначенных для работы с графикой, расстояние задается в единицах PCL. Единица PCL (PCL Unit) после выполнения команды инициализации по умолчанию принимает значение, равное 1/300 дюйма. Команда установки разрешения печати на величину единицы PCL не алияет.

Две команды позиционирования язык PCL позволяет объединить в одну следующим образом:

```
<Esc>, '*', 'p', '6', '0', '0', 'x' '3', '0', '0', 'Y'
```

Следующая по порядку операция — выбор разрешения для печати изображения. Например, включить режим с разрешением 300 точек/дюйм можно командой:

```
<Esc>, '*', 't', '3', '0', '0', 'R'
```

ПРИМЕЧАНИЕ

В принтерах HP используются квадратные пикселы. Разрешение по осям X и Y одинаковое, поэтому команда установки имеет только один параметр.

Команда установки разрешения печати воздействует не только на текущую, но и на все последующие страницы, вплоть до поступления команды сброса или другой команды установки разрешения.

От выбора разрешения сильно зависят качество и скорость печати. При увеличении разрешения качество изображения улучшается, но объем передаваемой информации возрастает в квадрате, а скорость вывода информации уменьшается пропорционально росту ее объема. Даже для бытовых лазерных принтеров, способных выводить на печать по 6–12 страниц в минуту, параллельный порт LPT в стан-

дартном режиме SPP работает слишком медленно и при использовании высокого разрешения может заметно притормаживать процесс печати, поэтому при выводе растровой графики желательно использовать режим ECP.

Разрешение 600 точек/дюйм можно задавать для лазерных принтеров и для старших (начиная с 6XX) моделей струйных принтеров HP при печати в черно-белом растровом режиме. Младшие модели струйных принтеров (до 5XX) могут работать с разрешением не более 300 точек/дюйм.

ПРИМЕЧАНИЕ

Команда установки разрешения автоматически переключает принтер в графический режим. По умолчанию печать графики начинается с текущей строки и левого края страницы, компрессия растра не используется.

Цветные принтеры имеют несколько режимов печати:

- черно-белый (однокомпонентный) режим;
- цветной трехкомпонентный режим RGB (красный, зеленый, синий);
- цветной трехкомпонентный режим CMY (бирюзовый, пурпурный, желтый);
- цветной четырехкомпонентный режим KCMY (черный, бирюзовый, пурпурный, желтый).

ПРИМЕЧАНИЕ

Четырехкомпонентный режим поддерживается только струйными принтерами, в которые устанааливаются сразу два картриджа (черный и цветной).

После сброса по умолчанию установлен черно-белый режим. Чтобы переключиться в другой режим, используется команда «Задать число цветовых плоскостей». Например, включить цветной режим CMY можно командой:

```
<Esc>, '*', 'r', '-', '3', 'U'
```

Поскольку каждый передаваемый байт содержит 8 точек изображения, при разделении строки растра на байты последний байт используется не полностью, если длина строки не кратна 8. Если программист сам формирует изображение, то неиспользуемые младшие разряды просто заполняются нулями, но при печати готового изображения из файла лучше задать ширину изображения в явном виде, чтобы не обрабатывать неполные байты. Например, чтобы установить ширину строки изображения 213 точек, нужно подать команду:

```
<Esc>, '*', 'r', '2', '1', '3', 'S'
```

Чтобы использовать отступ от левого края страницы, который был задан командой позиционирования курсора по горизонтали, следует подать команду, устанавливающую режим печати графического изображения с текущей позиции курсора:

<Esc>, '*', 'r', '1', 'A'

ПРИМЕЧАНИЕ

Перед началом вывода изображения обязательно должна быть подана по крайней мере одна из команд: «Установить разрешение» или «Начать печать графики». Рекомендуется, однако, подавать команду «Начать печать графики» независимо от того, была ли подана команда «Установить разрешение».

Обычно при печати используется принятый по умолчанию режим передачи данных без компрессии (режим 0), однако режим компрессии можно изменить перед началом вывода изображения и даже в процессе вывода (перед началом передачи очередной строки). Например, чтобы включить режим последовательного кодирования, нужно подать команду:

<Esc>, '*', 'b', '1', 'M'

После того, как нужный режим печати установлен, можно приступать к построчному выводу изображения. Для печати в монохромном режиме используется команда «Передать строку растра». В результате выполнения этой команды будет загружена одна строка растра, после чего курсор перемещается в начало следующей строки. Печать изображения на лазерном принтере будет отложена до полного завершения формирования образа страницы, а на струйном — до заполнения такого количества строк, которое соответствует количеству используемых в данном режиме сопел печатающей головки.

Команда «Передать строку растра» содержит размер строки в байтах и соответствующее количество байтов данных. К сожалению, размер указывается в кодировке ASCII, что может потребовать введения в программу дополнительных операций для преобразования двоичного числа в ASCII-код, если размер строки является переменной, а не константой. Например, чтобы распечатать с экрана черно-белое изображение шириной 640 пикселов по горизонтали, нужно задавать размер строки 80 байт (640/8):

<Esc>, '*', 'b', '8', '0', 'W', <80 байт данных>

Если при формировании изображения появляются чистые (белые) строки растра, для ускорения передачи желательно их пропустить.

Использовать для этой цели команду позиционирования курсора (по оси Y) на струйных принтерах не рекомендуется: вместо ускорения может получиться существенное замедление печати.

Вместо команды позиционирования курсора для пропуска пустой строки можно использовать команду «Передать строку растра» с нулевым количеством передаваемых байтов данных:

```
<Esc>, '*', 'b', '0', 'W'
```

Для цветной печати, кроме команды «Передать строку растра», используется также аналогичная ей по формату команда «Передать строку цветовой плоскости», которая после передачи данных выполняет переключение на следующую цветовую плоскость, а не на следующую строку растра.

Порядок передачи цветовых компонентов соответствует мнемоническому обозначению режима:

- в режиме RGB первой передается строка красной цветовой плоскости, затем — зеленой, затем — синей;
- в режиме CMY первой передается строка бирюзовой плоскости, затем — пурпурной, затем — желтой;
- в режиме KCMY первой передается строка черной плоскости, затем — бирюзовой, затем — пурпурной, затем — желтой.

Для передачи всех цветовых компонентов растровой строки, кроме последнего, используются следующие друг за другом команды «Передать строку цветовой плоскости»; последний компонент передается по команде «Передать строку растра».

ПРИМЕЧАНИЕ

Поступление команды «Передать строку растра» переводит курсор на следующую строку растра, а все оставшиеся незаполненными цветовые компоненты текущей строки обнуляются.

Например, для передачи одной цветной строки шириной 640 пикселов в формате CMY нужно последовательно подать три команды:

```
<Esc>, '*', 'b', '8', '0', 'V', <80 байт данных>
<Esc>, '*', 'b', '8', '0', 'V', <80 байт данных>
<Esc>, '*', 'b', '8', '0', 'W', <80 байт данных>
```

Пустые строки цветовых плоскостей можно пропускать, подавая команду передачи строки плоскости с нулевым значением количества байтов данных:

```
<Esc>, '*', 'b', '0', 'V'
```

Для пропуска пустых (белых) строк можно, как и в черно-белом режиме, использовать команду:

<Esc>, '*', 'b', '0', 'W'

После завершения вывода изображения нужно подать команду выхода из режима растровой печати:

<Esc>, '*', 'r', 'C'

Чтобы извлечь из принтера отпечатанную страницу, нужно послать команду перевода формата:

<FF>

В листинге 7.9 приведена программа Test_HP_On_LPT1, предназначенная для проверки совместимости струйных и лазерных принтеров с набором команд PCL Hewlett-Packard. Программа осуществляет печать графического изображения набора символов шрифта 8×16 в растровом режиме, используя для этого универсальные процедуры ввода-вывода из главы 1 «Работа с клавиатурой», а также процедуры из листингов 7.1 и 7.4.

Листинг 7.9. Тест для лазерных и струйных принтеров на совместимость с растровым режимом печати HP

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

DATASEG
; Номер печатаемой строки изображения
PrintingString DW ?
; Номер печатаемого байта
PrintingByte DW ?
; Текстовые сообщения
Txt1 DB LIGHTCYAN,0,19
     DB "ПЕЧАТЬ КОДОВОЙ ТАБЛИЦЫ РУССКОГО ШРИФТА DOS",0
     DB LIGHTCYAN,2,22
     DB "НА ЛАЗЕРНОМ HP-СОВМЕСТИМОМ ПРИНТЕРЕ",0
```

```
     DB LIGHTGREEN,12,11
     DB "Включите принтер, вставьте "
     DB "бумагу, установите режим ON-LINE",0
     DB LIGHTGREEN,14,17
     DB "(будет произведен переход в графический режим)",0
     DB YELLOW,24,14,"Нажмите любую клавишу и "
     DB "ждите завершения печати",0
Txt2 DB LIGHTGREEN,12,28,"Печать шрифта завершена",0
     DB YELLOW,24,29,"Нажмите любую клавишу",0

; КОМАНДЫ ДЛЯ ПРИНТЕРА
; Инициализировать принтер
PrinterReset     DB 2, 1Bh,'E'
; Установка разрешения 300 точек на дюйм
SetPrintDensity  DB 7, 1Bh,"*t300R"
; Задать длину строки 40 байт (320/8)
SetPLineLength   DB 6, 1Bh,"*b40W"
; Завершение работы в растровом режиме
ExitRasterMode   DB 4, 1Bh,"*rC"
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC Test_HP_On_LPT1
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Считать шрифт из видеопамяти
        call    GrabRusFont
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести текстовые сообщения на экран
        MShowColorText 5,Txt1
        call    GetChar
; Установить видеорежим VGA 320x200, 256 цветов
        mov     AX,13h
        int     10h
; Отобразить шрифт
        call    ShowRusFont
; Инициализировать принтер
        mov     SI,offset PrinterReset
        call    OutCommandToLPT1
; Установка разрешения 300 точек на дюйм
```

продолжение ⇒

Листинг 7.9 (*продолжение*)

```
        mov     SI,offset SetPrintDensity
        call    OutCommandToLPT1
; Настроить пару регистров ES:SI на видеопамять
        mov     AX,0A000h
        mov     ES,AX
        xor     SI,SI  ;обнулить SI
; Сбросить счетчик строк
        mov     [PrintingString],0

; ОСНОВНОЙ ЦИКЛ (ПО ПЕЧАТАЕМЫМ СТРОКАМ)
; Печать осуществляется в инверсной форме (светлые точки
; экрана при печати отображаются черными и наоборот).
; Вывод изображения на принтер выполняется по строкам
; растра, слева направо, сверху вниз.
@@P0:
; Задать длину строки 40 байт (320/8)
        push    SI
        mov     SI,offset SetPLineLength
        call    OutCommandToLPT1
        pop     SI
; Сбросить счетчик байтов
        mov     [PrintingByte],0
; Цикл по печатаемым байтам
@@P1:   mov     CX,8   ;счетчик точек в байте
        xor     AL,AL  ;обнулить байт
; Цикл по печатаеиым точкам
@@P2:   shl     AL,1   ;сдвинуть разряды влево
        cmp     [byte ptr ES:SI],0 ;цвет точки?
        je      @@P3   ;пропустить черную точку
        or      AL,1   ;"поставить" точку
@@P3:   ; Перейти на следующую точку
        inc     SI
        loop    @@P2
        ; Вывести байт на принтер
        call    OutCharToLPT1
        ; Перейти к следующему байту
        inc     [PrintingByte]
        cmp     [PrintingByte],40
        jl      @@P1
; Перейти на следующую строку экранного изображения
        inc     [PrintingString]
        cmp     [PrintingString],160
        jl      @@P0

; Завершение работы в растровом режиме
        mov     SI,offset ExitRasterMode
        call    OutCommandToLPT1
; Послать на принтер код завершения страницы
        mov     AL,0Ch
```

```
        call    OutCharToLPT1
; Переустановить текстовый режим
        mov     ax,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        call    SetCursorPosition
; Вывести сообщение о завершении записи шрифта
        MShowColorString Txt1
        MShowColorText 2,Txt2
        call    GetChar
@@End:  ; Переустановить текстовый режим
        mov     ax,3
        int     10h
        ; Выход в DOS
        mov     AH,4Ch
        int     21h
ENDP Test_HP_On_LPT1
ENDS

; Подключить процедуры вывода данных на экран
include "list1_02.inc"
; Подключить процедуры вывода символа и посылки
; команды на принтер
include "list7_01.inc"
; Подключить процедуры для захвата русского шрифта и
; отображения его в режиме 320x200
include "list7_04.inc"

END
```

ПРИМЕЧАНИЕ

Для запуска теста пригоден любой AT-совместимый компьютер, оснащенный струйным или лазерным принтером, подключенным к порту LPT1.

СОВЕТ

Прежде чем запустить тест, извлеките пачку листов из лотка подачи бумаги и оставьте в нем только один лист, а лучше вообще переключите принтер на ручную подачу.

Глава 8
Шина USB

За последние десять лет появилось множество новых разновидностей периферийного оборудования, что породило проблему одновременного подключения большого количества устройств к системному блоку персонального компьютера. Блок в этом случае должен иметь десятки разъемов, из которых к устройствам тянется соответствующее количество кабелей, а каждое устройство имеет, помимо интерфейсного кабеля, также шнур питания.

В офисе устройства обычно закреплены на определенных местах, а кабели прокладываются по стенам специалистами-монтажниками, поэтому в первую очередь новая проблема проявила себя в бытовой сфере: при использовании традиционного способа подключения оборудования персональный компьютер оказывается опутанным множеством проводов, которые скручиваются и переплетаются при перемещении устройств с места на место.

Для решения проблемы соединительных кабелей разработчики аппаратуры стали использовать высокоскоростные последовательные каналы передачи данных и кабельные концентраторы (хабы). В сигнальные кабели добавлены дополнительные провода, обеспечивающие подачу питания на устройства с небольшим энергопотреблением от хабов, что позволяет значительно сократить количество силовых кабелей в системе.

Универсальная последовательная шина (Universal Serial Bus, сокращенно USB) разрабатывалась как промышленный стандарт расширения архитектуры PC, ориентированный на интеграцию с устройствами телефонии и бытовой электроники. Разработка шины ведется с 1994 года. Первоначально в группу разработчиков входили Compaq, DEC, IBM, Intel, Microsoft, NEC и Northern Telecom, а затем количество заинтересованных участников стало

расширяться. Первая официальная версия стандарта на шину USB (спецификация 1.0) была опубликована в 1996 году, доработанный вариант (спецификация 1.1) [95] появился в 1998 году, а в 2000 году опубликована вторая версия стандарта (спецификация 2.0). К сожалению, использование последней версии стандарта в течение длительного времени тормозилось, в том числе по вине самих разработчиков: устройства с интерфейсом USB 2.0 не выпускались потому, что для них отсутствовало системное программное обеспечение, которое не удавалось отладить потому, что не выпускались устройства!

Так как устройства, соответствующие спецификации USB 2.0, до сих пор мало распространены, ниже мы будем рассматривать устаревшую, но широко используемую в данный момент версию 1.1. Подробную документацию по всем версиям стандарта можно найти в Интернете, на сайте USB-консорциума www.usb.org.

Архитектура шины USB

Архитектура и основные параметры шины USB определяются возложенными на нее задачами. Физическая топология шины USB, изображенная на рис. 8.1, имеет следующие основные особенности:

- шина обеспечивает подключение USB-устройств к хосту USB;
- физическое соединение устройств между собой осуществляется по топологии многоярусной звезды;
- центром каждой звезды является хаб;
- каждый кабельный сегмент соединяет между собой две точки: хост с хабом или функцией, хаб с функцией или другим хабом.

Хост-контроллер (Host Controller) – это главный контроллер, который входит в состав системного блока компьютера и управляет работой всех устройств на шине USB.

На шине USB допускается наличие только одного хоста. Системный блок AT-совместимого персонального компьютера может содержать от одного до трех хост-контроллеров [80], каждый из которых управляет отдельной шиной USB.

Устройство (Device) USB может быть хабом или функцией.

Хаб (Hub) – это устройство, которое обеспечивает дополнительные точки подключения к шине USB.

Каждый хаб имеет один **восходящий порт** (Upstream Port), предназначенный для подключения к хабу верхнего уровня, и несколько

нисходящих портов (Downstream Ports), предназначенных для подключения функций или хабов нижнего уровня.

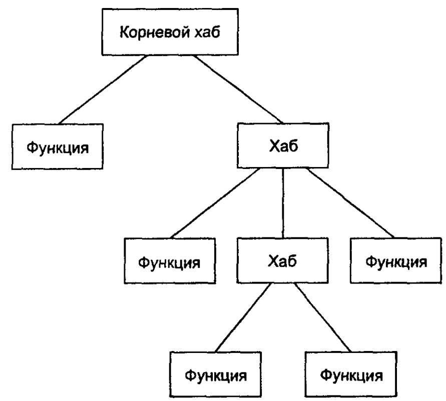


Рис. 8.1. Физическая топология шины USB

Хаб может иметь собственный источник питания (Self-powered Hub) или получать питание от шины USB (Bus-powered Hub). Хаб управляет работой нисходящих портов, осуществляет контроль подключения и отключения устройств. Через порты хаб управляет электропитанием устройсв, не имеющих собсвенных источников энергии.

Корневой хаб (Root Hub) — это хаб, который входит в состав хост-контроллера.

Функция (Function) — это периферийное устройство или отдельный блок периферийного устройства, способный передавать и принимать информацию по шине USB.

Составное устройство (Compound Device) — это периферийное устройство со встроенным хабом.

Различают три уровня взаимодействия хоста с физическим устройством:

- на верхнем уровне (уровне функции) между собой взаимодействуют клиентская программа и функция;
- на среднем уровне (уровне устройства) взаимодействуют системное программное обеспечение и логическое устройство USB;

- на нижнем уровне (уровне интерфейса шины USB) хост-контроллер взаимодействует с USB-интерфейсом устройства.

Используемая на среднем уровне взаимодействия логическая топология шины USB (рис. 8.2) гораздо проще физической: хост обменивается информацией с логическими устройствами таким образом, что точка подключения устройства не имеет значения, как если бы все устройства были подключены к корневому хабу. Верхний уровень взаимодействия вопросы топологии шины вообще не затрагивает.

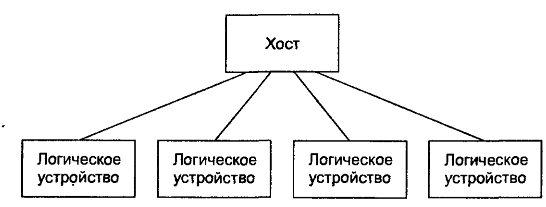


Рис. 8.2. Логическая топология шины USB

Режимы передачи данных

Пропускная способность шины USB, соответствующей спецификации 1.1, составляет 12 Мбит/с (1,5 Мбайт/с). Полоса пропускания шины делится между всеми устройствами, подключенными к шине.

Шина USB имеет два режима передачи: в полноскоростном (full-speed) режиме скорость передачи составляет 12 Мбит/с, в низкоскоростном (low-speed) — 1,5 Мбит/с. Полноскоростной режим используется принтерами, сканерами, видеокамерами и другими устройствами, передающими больше объемы информации. Низкоскоростной режим предназначен для упрощения конструкции и снижения себестоимости устройств, обменивающихся с компьютером небольшими порциями данных — мыши, джойстика и т. п.

Устройство, использующее полную скорость передачи, именуется полноскоростным (Full-speed Device), а устройство, использующее пониженную скорость — низкоскоростным (Low-speed Device).

Модель передачи данных

Логическое устройство USB представляет собой набор конечных точек.

Конечная точка (Endpoint) — это часть устройства USB, которая имеет уникальный идентификатор и является получателем или отправителем информации, передаваемой по шине USB.

Основными для конечной точки являются следующие параметры:

- частота доступа к шине;
- допустимая величина задержки обслуживания;
- требуемая ширина полосы пропускания канала;
- номер конечной точки;
- способ обработки ошибок;
- максимальный размер пакетов, которые конечная точка может принимать или отправлять;
- используемый конечной точкой тип посылок;
- направление передачи данных.

Любое USB-устройство имеет конечную точку с нулевым номером, которая в документации именуется Endpoint Zero или Endpoint 0. Нулевая точка позволяет хосту опрашивать устройство с целью определения его типа и параметров, выполнять инициализацию и конфигурирование устройства.

Каждая конечная точка может работать только с одним типом посылок (типы посылок описаны ниже). Нулевая точка поддерживает управляющие посылки и поэтому может выполнять как прием, так и передачу данных.

Кроме нулевой точки, функции обычно имеют дополнительные конечные точки, которые используются для обмена данными с хостом. Дополнительные точки могут работать либо только на прием, либо только на передачу информации. Точки, передающие данные хосту, именуются **входными** (IN), а точки, принимающие данные от хоста — **выходными** (OUT). Низкоскоростные устройства могут иметь до двух дополнительных точек, а полноскоростные — до 15 дополнительных входных и до 15 дополнительных выходных точек.

Хост получает доступ к нулевой конечной точке после того, как устройство присоединено к шине, включено и получило сигнал сброса по шине. Все остальные конечные точки, кроме точки с нулевым номером, после включения питания или сброса устройства находятся в неопределенном состоянии и недоступны для работы до тех пор, пока хост не выполнит процедуру конфигурирования устройства.

Для описания порядка прохождения информации через буфер данных между программным обеспечением и конечной точкой в спецификации USB введено понятие канала.

Канал (pipe) — это модель взаимодействия конечной точки с программным обеспечением хоста. Имеется два типа каналов:

- **поток** (stream) — это канал для передачи данных, структура которых определяется клиентским программным обеспечением. Потоки используются для передачи массивов данных, передачи данных по прерываниям и изохронной передачи данных.
- **канал сообщений** (message pipe) — это канал для передачи данных, структура которых определяется спецификацией на шиину USB. Каналы сообщений применяются для передачи управляющих посылок.

Основными характеристиками каналов являются:

- полоса пропускания канала;
- используемый каналом тип передачи данных;
- характеристики, соответствующие конечной точке: направление передачи данных и максимальный размер пакета.

Канал сообщений, связанный с нулевой конечной точкой, называется в документации Основным каналом сообщений (Default Control Pipe). Спецификация USB запрещает для канала сообщений одновременную обработку нескольких запросов: нельзя начинать передачу нового сообщения, пока не завершена обработка предыдущего. В случае возникновения ошибки, однако, передача сообщения может быть прервана хостом, после чего хост может начать передачу нового сообщения.

Структура пакетов

Вся информация передается по шине USB в виде пакетов. Каждый пакет начинается с поля синхронизации (SYNC), за которым следует идентификатор пакета (PID). Идентификатор пакета состоит из четырехразрядного кода типа пакета и четырехразрядного контрольного поля, каждый разряд которого является инверсией соответствующего разряда кода типа пакета (код пакета и контрольное поле комплементарны). Принятые по стандарту USB 1.1 коды PID перечислены в табл. 8.1. Как видно из таблицы, имеется четыре группы идентификаторов, причем принадлежность к определенной группе задается в двух младших разрядах PID:

- 00b — специальный пакет (Special),
- 01b — маркер (Token),
- 10b — подтверждение (Handshake),
- 11b — пакет данных (Data).

Таблица 8.1. Список кодов PID

Тип PID	Обозначение	Код типа	Описание пакета
Маркер	OUT	0001b	Адрес и номер конечной точки при передаче от хоста к функции
	IN	1001b	Адрес и номер конечной точки при передаче от функции к хосту
	SOF	0101b	Маркер начала кадра и номер кадра
	SETUP	1101b	Адрес и номер конечной точки при передаче команды от хоста к функции
Данные	DATA0	0011b	Четный пакет данных
	DATA1	1011b	Нечетный пакет данных
Подтверждение	ACK	0010b	Подтверждение приема пакета
	NAK	1010b	Ответ на запрос не готов
	STALL	1110b	Произошел сбой в конечной точке или запрос не поддерживается
Специальный	PRE	1100b	Преамбула запроса, которая разрешает замедленный трафик для низкоскоростных устройств

Структура пакета зависит от группы, к которой он относится.

- Маркер начала кадра содержит 8-разрядное поле PID, 11-разрядный номер кадра и 5-разрядный циклический контрольный код. Маркер начала кадра, как следует из его названия, отмечает начало каждого нового кадра на шине USB.
- Маркер транзакции содержит 8-разрядное поле PID, 7-разрядное поле адреса функции, 4-разрядное поле адреса конечной точки и 5-разрядный циклический контрольный код. Маркер транзакции отмечает начало очередной транзакции на шине USB.
- Пакет данных начинается с 8-разрядного поля PID, за которым следует от 0 до 1023 байт данных и 16-разрядный циклический контрольный код.
- Подтверждение содержит только 8-разрядное поле PID. Подтверждение завершает каждую транзакцию.

Порядок выполнения транзакций

Все транзакции на шине USB выполняются под управлением хост-контроллера. Передача данных возможна только по запросам хоста:

периферийные устройства не могут выдать на шину какую-либо информацию по собственной инициативе, не могут самостоятельно посылать запросы прерываний.

Разные способы передачи отличаются друг от друга своими характеристиками, к которым относятся:

- формат данных;
- направление передачи данных;
- ограничения на размер пакета;
- ограничения на доступ к шине,
- допустимая величина задержки обслуживания;
- требуемая последовательность передачи данных;
- способ обработки ошибок.

Различают следующие типы транзакций:

- *передача команды*: хост передает заданной конечной точке код команды;
- *изохронная передача данных*: хост передает заданной конечной точке блок данных, не ожидая подтверждения (при сбоях операция не повторяется);
- *передача данных с подтверждением*: хост передает заданной конечной точке блок данных и ожидает подтверждение приема (при сбоях операция повторяется, время выполнения не ограничено);
- *изохронный прием данных*: заданная конечная точка передает хосту блок данных, не ожидая подтверждения (при сбоях операция не повторяется);
- *прием данных с подтверждением*: заданная конечная точка передает хосту блок данных и ожидает подтверждения приема (при сбоях операция повторяется, время выполнения не ограничено).

Транзакция передачи команды включает в себя следующие операции:

- хост посылает маркер SETUP, содержащий номер функции и номер конечной точки, для которой предназначена команда;
- хост посылает выбранной конечной точке пакет данных со сброшенным битом синхронизации (DATA0), содержащий 8-байтный код команды;
- функция посылает хосту пакет подтверждения.

Транзакция передачи данных с подтверждением включает следующие операции:

- хост посылает маркер OUT, содержащий номер функции и номер конечной точки, для которой предназначены данные;
- хост посылает выбранной конечной точке пакет данных;
- функция посылает хосту пакет подтверждения.

Транзакция приема данных с подтверждением включает следующие операции:

- хост посылает маркер IN, содержащий номер функции и номер конечной точки, от которой запрашиваются данные;
- выбранная конечная точка передает хосту пакет данных или пакет подтверждения (NAK — данные не готовы, STALL — сбой)
- если хост получил пакет данных, он посылает пакет подтверждения (ACK).

Транзакция изохронной передачи данных включает следующие операции:

- хост посылает маркер OUT, содержащий номер функции и номер конечной точки, для которой предназначены данные;
- хост посылает выбранной конечной точке пакет данных со сброшенным битом синхронизации (DATA0).

Транзакция изохронного приема данных включает следующие операции:

- хост посылает маркер IN, содержащий номер функции и номер конечной точки, от которой запрашиваются данные;
- выбранная конечная точка передает хосту пакет данных со сброшенным битом синхронизации (DATA0).

При выполнении транзакций используется три типа пакетов подтверждения:

- ACK — информация принята получателем без ошибок, операция успешно завершена;
- NAK — функция занята (не готова к приему или передаче данных);
- STALL — произошел сбой при выполнении операции, функция не может принять или передать данные.

Если при выполнении транзакции передачи данных с подтверждением в пакете данных обнаружена ошибка по контрольному коду CRC, получатель пакета данных не высылает пакет подтверждения. Отправитель при отсутствии подтверждения от получателя должен зафиксировать ошибку передачи данных и повторить транзакцию.

Типы посылок

Архитектура USB предусматривает четыре типа посылок.

- *Управляющие посылки* (Control Transfers) используются хост-контроллером конфигурирования устройства. Процесс конфигурирования предусматривает опрос устройства (с целью получения информации о типе и свойствах устройства) и настройку параметров устройства на заданный режим работы.
- *Передачи массивов данных* (Bulk Data Transfers) применяются, когда требуется обеспечить гарантированную доставку данных от хоста к функции или от функции к хосту, но время доставки не ограничено. Передачи массивов данных характерны для принтеров и сканеров.
- *Передачи по прерываниям* (Interrupt Transfers) используются в том случае, когда требуется передавать одиночные пакеты данных небольшого размера, каждый пакет требуется гарантированно передать за ограниченное время, а операции передачи носят спонтанный (случайный) характер. Передача по прерываниям характерна для клавиатуры и координатных устройств: джойстика, мыши и т. п.
- *Изохронные передачи* (Isochronous Transfers) применяются для обмена данными в «реальном времени», когда на каждом временном интервале требуется передавать строго определенное количество данных, но доставка информации не гарантирована (передача данных ведется без повторения при сбоях, допускается потеря пакетов). Изохронные передачи обычно применяются мультимедийными устройствах для передачи аудио- и видеоданных.

Порядок передачи управляющих посылок

Есть три типа управляющих посылок:

- посылка записи данных (Control Write);
- посылка чтения данных (Control Read);
- посылка без данных (No-data Control).

Управляющая посылка записи данных включает следующие транзакции:

- передача команды;
- передача (с подтверждением) одного или нескольких пакетов данных;

- прием (с подтверждением) пустого пакета данных, подтверждающего успешное завершение операции.

Управляющая посылка чтения данных включает следующие транзакции:

- передача команды;
- прием (с подтверждением) одного или нескольких пакетов данных;
- передача (с подтверждением) пустого пакета данных, подтверждающего успешное завершение операции.

Управляющая посылка без данных включает следующие транзакции:

- передача команды;
- прием (с подтверждением) пустого пакета данных, подтверждающего успешное завершение операции.

При выполнении транзакции передачи команды признак синхронизации данных должен быть сброшен в ноль (блок данных, содержащий код команды, имеет PID DATA0).

Если команда предполагает приём или передачу данных, то после каждой транзакции признак синхронизации данных инвертируется: первый блок данных имеет идентификатор DATA1, второй — DATA0, третий — DATA1 и т. д.

Пустой пакет данных, подтверждающий завершение управляющей посылки, должен иметь идентификатор DATA1.

При передаче управляющей посылки максимальный размер пакета для полноскоростного устройства может составлять 8, 16, 32 или 64 байта, а для низкоскоростного всегда равен 8 байтам.

ПРИМЕЧАНИЕ

На практике для передачи сообщений по Основному каналу сообщений всегда используется максимальный размер пакета, равный 8 байтам.

Порядок передачи массивов данных

Различают два вида передачи массивов:

- передача массива данных от хоста к конечной точке (Bulk Write);
- прием хостом массива данных от конечной точки (Bulk Read).

Передача данных от хоста к конечной точке состоит из следующих друг за другом транзакций передачи данных с подтверждением, а передача данных от конечной точки к хосту — из следующих друг

за другом транзакций приема данных с подтверждением. И в том, и в другом случае перед началом передачи массива триггер синхронизации данных должен быть сброшен в 0: при выполнении первой транзакции блок данных имеет идентификатор DATA0, второй — DATA1, третий — DATA0 и т. д.

Прием и передачу массивов данных могут выполнять только полноскоростные устройства. Максимальный размер пакета при передаче массива может быть равен 8, 16, 32 или 64 байтам (обычно используется значение 64 байта).

Порядок передачи данных по прерываниям

Различают два вида передачи по прерываниям:

- передача данных от хоста к конечной точке по прерыванию;
- прием данных хостом от конечной точки по прерыванию.

Передача данных по прерыванию заключается в выполнении транзакции передачи пакета данных (с подтверждением) от хоста к конечной точке. Прием данных заключается в выполнении транзакции приема пакета данных (с подтверждением) от конечной точки.

При приеме или передаче каждого блока данных происходит переключение триггера данных. Первый передаваемый (или принимаемый) блок имеет идентификатор DATA0, следующий — DATA1 и т. д.

Максимальный размер пакета при передаче по прерываниям для низкоскоростного устройства не может быть более 8 байт, для полноскоростного — более 64 байт.

Порядок выполнения изохронной передачи

Различают два вида изохронной передачи:

- изохронная передача данных от хоста к конечной точке;
- изохронный прием данных хостом от конечной точки.

Изохронная передача данных заключается в выполнении транзакции передачи пакета данных (без подтверждением) от хоста к конечной точке. Изохронный прием данных заключается в выполнении транзакции приема пакета данных (без подтверждения) от конечной точки.

Состояние триггера данных при изохронной передаче игнорируется, но рекомендуется сбросить его в ноль перед началом передачи.

Изохронную передачу могут выполнять только полноскоростные устройства. Максимальный размер пакета данных при изохронной передаче — 1023 байта.

Структура кадра USB

Длительность каждого кадра USB по времени равна одной миллисекунде. Хост-контроллер в начале каждого кадра генерирует маркер начала кадра, после чего начинает выполнять передачу данных. Передачи внутри кадра выполняются в следующем порядке:

- изохронные передачи;
- передачи по прерываниям;
- передачи управляющих посылок;
- передачи массивов данных.

Таким образом, изохронные передачи имеют высший приоритет, а передачи массивов данных — самый низкий.

Регистры хост-контроллера

Драйвер интерфейса USB управляет работой хост-контроллера через регистры. Регистры универсального хост-контроллера принято разделять на две группы: группу конфигурационных регистров PCI (USB PCI Configuration Registers) и группу регистров пространства ввода-вывода (USB Host Controller IO Space Registers). Ниже мы будем рассматривать только регистры ввода-вывода, так как непосредственная работа с конфигурационными регистрами из прикладных программ нежелательна (может привести к «зависанию» системы).

Для описания режима доступа к данным в регистрах USB используются следующие стандартные обозначения:

- **RO** — возможно только считывание данных;
- **WO** — возможна только запись данных;
- **R/W** — разрешено выполнение как записи, так и считывания данных;
- **R/WC** — разрешено считывание данных и сброс отдельных разрядов регистра (запись единицы в некоторый разряд регистра приводит к тому, что этот разряд сбрасывается в ноль).

Список регистров ввода-вывода хост-контроллера шины USB приведен в табл. 8.2. Доступ к этим регистрам осуществляется через группу портов ввода/вывода, базовый адрес которой задан в конфигурационном регистре USBBA.

Таблица 8.2. Регистры ввода-вывода универсального хост-контроллера шины USB

Смещение	Размер	Доступ	Мнемоника	Наименование регистра
00h	WORD	R/W	USBCMD	Регистр команды USB
02h	WORD	R/WC	USBSTS	Регистр состояния USB
04h	WORD	R/W	USBINTR	Регистр управления прерываниями USB
06h	WORD	R/W	FRNUM	Регистр номера кадра USB
08h	DWORD	R/W	FLBASEADD	Регистр базового адреса списка кадров USB
0Ch	BYTE	R/W	SOFTMOD	Регистр модификатора начала кадра USB
10h	WORD	R/WC	PORTSC0	Регистр состояния и управления порта 0
12h	WORD	R/WC	PORTSC1	Регистр состояния и управления порта 1

Регистр команды USB (USBCMD) предназначен для передачи команд хост-контроллеру и доступен как для записи, так и для считывания данных. Контроллер начинает выполнение команды сразу же после того, как она записана в регистр.

Рассмотрим назначение разрядов регистра команды USB.

- Бит 0 (RS) — запуск/останов. Запись единицы в данный разряд активизирует работу контроллера (контроллер приступает к обработке и передаче данных), а запись нуля приводит к немедленной остановке контроллера и прекращению всех выполняемых операций. Контроллер сам может сбрасывать данный разряд в ноль в случае возникновения серьезных ошибок и сбоев.
- Бит 1 (HCRESET) — сброс хост-контроллера. Запись единицы в данный разряд приводит к сбросу регистров, отражающих внутреннее состояние контроллера: механизм, обеспечивающий обнаружение подсоединения и отсоединения устройств обнуляется, работа обоих портов хост-контроллера блокируется. В результате происходит «виртуальное отсоединение» подключенных к контроллеру устройств, биты 1 и 3 в регистрах состояния портов контроллера устанавливаются в единицу, а биты 0 и 8 сбрасываются. После завершения процесса обнуления всех внутренних регистров контроллер *самостоятельно* сбрасывает бит HCRESET и разрешает обнаружение подсоединенных устройств и определение скорости их работы, что приводит к соответствующему изменению битов 0 и 8 в регистрах состояния портов.

- Бит 2 (GRESET) — глобальный сброс. Запись единицы в данный разряд вызывает общий сброс хост-контроллера и всех подключенных к нему устройств. Снять сигнал глобального сброса можно по прошествии не менее 10 мс после его установки, записав в данный разряд ноль.
- Бит 3 (EGSM) — переключение в глобальный режим ожидания. Запись единицы в данный разряд вызывает переключение хост-контроллера и всех подключенных к нему устройств в режим ожидания. Перед установкой в единицу бита EGSM необходимо остановить контроллер, сбросив бит запуска/останова RS. При выходе из режима ожидания данный бит сбрасывается в ноль программным обеспечением после сброса в ноль бита 4.
- Бит 4 (FGR) — общий выход из режима ожидания. Запись единицы в данный разряд выводит хост-контроллер и подключенные к нему устройства из режима ожидания. Устанавливать данный разряд может не только программное обеспечение, но и сам хост-контроллер — при обнаружении подключения или отключения устройства во время пребывания системы в режиме ожидания. Снять сигнал «пробуждения» можно по прошествии не менее 20 мс после его установки, записав в данный разряд ноль.
- Бит 5 (SWDBG) — переключение в режим отладки. Устанавливать бит SWDBG можно только при сброшенном бите RS, то есть только тогда, когда работа контроллера приостановлена. Запись единицы в данный разряд переводит контроллер в режим отладки программного обеспечения. В режиме отладки контроллер останавливается после выполнения каждой транзакции и сбрасывает бит RS; возобновление работы контроллера происходит после того, как программное обеспечение установит бит RS в единицу.
- Бит 6 (CF) — флаг завершения конфигурирования контроллера. Данный разряд может быть установлен в единицу программным обеспечением после завершения процесса конфигурирования хост-контроллера, но на работу самого контроллера никак не влияет.
- Бит 7 (MAXP) — максимальный размер пакета завершения кадра (0 — 32 байта, 1 — 64 байта).
- Биты 8–15 зарезервированы (всегда должны быть сброшены в ноль).

После аппаратного или программного сброса контроллера регистр команды USB содержит значение 0000h.

Регистр состояния USB (USBSTS) отражает текущее состояние хост-контроллера. Регистр USBSTS доступен для чтения и сброса (запись единиц в какие-либо его разряды сбрасывает эти разряды в ноль). Рассмотрим назначение разрядов регистра состояния USB:

- бит 0 (USBINT) — признак USB-прерывания. Данный разряд устанавливается контроллером при возникновении запроса прерывания по завершению транзакции (при установленном бите IOC в дескрипторе передачи) или при обнаружении короткого пакета (размер пакета меньше заданной в дескрипторе величины);
- бит 1 — признак прерывания по ошибке, которая произошла при выполнении транзакции;
- бит 2 (RSM_DET) — признак поступления на шину сигнала «пробуждения» от устройства USB;
- бит 3 — признак системной ошибки (устанавливается при возникновении сбоев в процессе передачи данных по шине PCI);
- бит 4 — признак обнаружения ошибки в работе контроллера;
- бит 5 — признак останова контроллера (устанавливается после сброса в ноль бита RS в регистре команды USB);
- биты 6–15 зарезервированы.

После аппаратного или программного сброса контроллера регистр состояния USB содержит значение 0020h.

Регистр управления прерываниями (USBINTR) позволяет разрешать и запрещать генерацию прерываний различных типов хост-контроллером. Регистр USBINTR доступен для записи и считывания. Назначение разрядов регистра управления прерываниями:

- бит 0 — управление прерыванием по тайм-ауту и обнаружению ошибок CRC (0 — прерывание запрещено, 1 — разрешено);
- бит 1 — управление прерыванием по сигналу пробуждения (0 — прерывание запрещено, 1 — разрешено);
- бит 2 — управление прерыванием по завершению транзакции IOC (0 — прерывание запрещено, 1 — разрешено);
- бит 3 — управление прерыванием по обнаружению короткого пакета (0 — прерывание запрещено, 1 — разрешено);
- биты 4–15 зарезервированы.

Таким образом, регистр управления прерываниями позволяет заблокировать любые прерывания от контроллера USB, кроме прерываний, генерируемых при обнаружении ошибок в работе самого контроллера.

После аппаратного или программного сброса контроллера регистр USBINTR содержит значение 0000h: все прерывания (за исключением прерываний по сбоям в работе контроллера) запрещены.

Регистр номера кадра (FRNUM) содержит текущий номер кадра USB. Младшие 11 разрядов регистра (биты 0–10) содержат текущий номер кадра, а остальные разряды зарезервированы и должны содержать нули. Регистр доступен для чтения в любой момент времени, а запись данных разрешена только в том случае, если работа контроллера приостановлена (бит RS в регистре команды USB сброшен в ноль).

Значение, содержащееся в разрядах 0–10 регистра FRNUM, увеличивается на единицу (инкрементируется) после завершения каждого кадра; после достижения значения 7FFh регистр обнуляется. Содержимое разрядов 0–10 служит номером кадра и передается в начале кадра в SOF-пакете. Кроме того, разряды 0–9 используются при формировании индекса текущего элемента в списке кадров (соответствуют разрядам 2–11 индекса).

После аппаратного или программного сброса контроллера регистр FRNUM содержит значение 0000h.

Регистр базового адреса списка кадров USB (FRBASEADD) содержит начальный (абсолютный) адрес списка кадров в оперативной памяти компьютера. Регистр доступен FRBASEADD для записи и считывания данных. Используются только старшие 20 бит регистра FRBASEADD, соответствующие битам 12–31 линейного адреса, а младшие 12 бит зарезервированы и должны содержать нули. Таким образом, базовый адрес списка кадров должен быть выровнен на границу гранулярности свопинга памяти процессоров Intel x86 (4 Кбайт).

Контроллер формирует указатель на текущий элемент списка кадров путем комбинирования сдвинутых влево на два разряда битов 0–9 из регистра номера кадра и битов 12–31 из регистра базового адреса. Разряды 0 и 1 указателя всегда равны нулю (указатель выровнен на границу двойного слова). Количество указателей в списке кадров равно 1024, а размер списка составляет 4 Кбайт.

После аппаратного или программного сброса контроллера значение регистра базового адреса считается «неопределенным»: перед запуском контроллера надлежит создать в оперативной памяти список кадров и загрузить его абсолютный (линейный) адрес в регистр FRBASEADD.

Регистр модификатора начала кадра USB (SOFMOD) служит для подстройки частоты кадров USB с целью обеспечения синхронизации

всех устройств системы при работе в режиме реального времени. Значение младших семи разрядов этого регистра складывается с числом 11936, в результате чего формируется делитель частоты кварцевого резонатора генератора тактовой частоты. Старший разряд регистра SOFMOD (бит 7) зарезервирован и должен содержать значение 0.

Частота кварцевого резонатора составляет 12 МГц, а значение, устанавливаемое в регистре модификатора начала кадра после аппаратного или программного сброса контроллера равно 64 (40h), поэтому частота генерации кадров равна 1 кГц. Изменяя значение модификатора от 0 до 127, можно осуществлять подстройку частоты кадров USB в пределах ± 0,5 %.

Регистр состояния и управления порта (PORTSC) позволяет контролировать режим работы порта хост-контроллера. Регистры PORTSC0 и PORTSC1 доступны для записи и считывания данных. Назначение разрядов регистра состояния порта:

- бит 0 — текущий статус подключения. Данный разряд доступен только для считывания и служит для определения наличия подключения USB-устройства к данному порту (0 — к порту ничего не подключено, 1 — к порту подключено устройство USB);
- бит 1 — признак изменения статуса подключения: устанавливается в единицу при любых изменениях текущего статуса подключения (см. бит 0). Бит признака изменения статуса подключения доступен для считывания и сброса (запись единицы в данный разряд сбрасывает его в ноль);
- бит 2 (PORT_EN) — включение и отключение порта (0 — порт заблокирован, 1 — работа порта разрешена). Данный разряд доступен для записи и считывания: запись нуля (блокировка порта) может выполняться как программным обеспечением, так и хост-контроллером (при возникновении сбоя в работе порта), а запись единицы — только программным обеспечением. Состояние данного разряда не изменяется, пока не изменится реальное состояние порта (возможна задержка срабатывания);
- бит 3 — признак включения и отключения порта (0 — состояние порта не изменялось, 1 — произошло включение или отключение порта). Бит признака включения и отключения порта доступен для считывания и сброса (запись единицы в данный разряд сбрасывает его в ноль);
- бит 4 — состояние линии D+. Данный бит отражает текущей логический уровень линии D+ и доступен только для чтения;

- бит 5 – состояние линии D–. Данный бит отражает текущей логический уровень линии D– и доступен только для чтения;
- бит 6 (RSM_DET) – признак обнаружения сигнала пробуждения (0 – сигнал не поступал, 1 – поступил сигнал пробуждения). Бит признака обнаружения сигнала пробуждения доступен для записи и считывания. Хост-контроллер устанавливает бит RSM_DET в 1 при обнаружении сигнала пробуждения; программное обеспечение устанавливает бит RSM_DET для формирования сигнала пробуждения. Если бит RSM_DET имеет значение 1, запись нуля приводит к тому, что порт посылает низкоскоростной EDP (до окончания EDP бит остается в состоянии 1);
- бит 7 зарезервирован, доступен только для считывания и при считывании всегда имеет значение 1;
- бит 8 – признак подключения низкоскоростного устройства. Данный бит доступен только для считывания и устанавливается в единицу, если к порту подключено низкоскоростное устройство;
- бит 9 – сброс порта. Бит сброса в современных контроллерах доступен только для считывания, хотя ранние варианты допускали выполнение операции записи. Бит сброса порта устанавливается в единицу при подаче команды сброса и находится в этом состоянии до тех пор, пока процедура сброса не будет завершена;
- бит 10 – признак активности линии «Overcurrent» (0 – линия неактивна, 1 – линия активна). Бит 10 используется лишь в современных контроллерах и доступен только для считывания;
- бит 11 – признак изменения состояния линии «Overcurrent». Данный бит доступен для считывания и сброса. Он устанавливается контроллером в состояние 1 при переходе линии «Overcurrent» из неактивного состояния в активное. Признак сбрасывается программным обеспечением путем записи единицы в бит 11. Бит 11 используется только в современных контроллерах;
- бит 12 – признак режима ожидания (устанавливается в 1, когда порт находится в режиме ожидания). Данный бит доступен для считывания и записи; он может использоваться программным обеспечением для перевода в состояние ожидания отдельного порта;
- биты 13–15 зарезервированы.

После аппаратного или глобального сброса оба регистра PDRTSC содержат значение 0080h: бит 7 имеет значение 1, остальные разряды сброшены. После сброса контроллера (HCRESET) могут быть установ-

лены биты 1 и 3. В современных контроллерах после осуществления сброса бит 11 может иметь значение 1.

Структуры данных хост-контроллера

Описание используемых хост-контроллером шины USB 1.1 структур данных содержится в спецификации Universal Host Controller Interface Design Guide [91].

Список кадров

Список кадров (Frame List) представляет собой массив, который состоит из 1024 указателей кадров по 32 разряда и занимает 4 Кбайт оперативной памяти. Начальный адрес списка хранится в регистре базового адреса списка кадров FLBASEADD (он должен быть выровнен на границу 4 Кбайт).

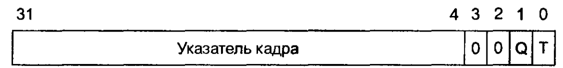


Рис. 8.3. Структура элемента списка кадров

Структура элемента списка кадров показана на рис. 8.3. Разряды элемента списка имеют следующее назначение:

- бит 0 (T) — признак «пустого» кадра (0 — указатель кадра является достоверным и содержит адрес заголовка очереди или дескриптора передачи, 1 — указатель не является достоверным и не должен обрабатываться);
- бит 1 (Q) — тип структуры данных, адрес которой содержится в указателе кадра (0 — дескриптор передачи, 1 — заголовок очереди);
- биты 2 и 3 зарезервированы и должны иметь значение 0;
- биты 4–31 (FLP) — биты 4–31 указателя кадра.

Указатель кадра (Frame List Pointer) содержит линейный (абсолютный) 32-разрядный адрес области памяти, по которому размещен объект данных списка кадров — заголовок очереди или дескриптор передачи.

Адрес объекта данных должен быть выровнен по границе 16 байт; младшие 4 разряда адреса всегда равны нулю.

Дескриптор передачи

Дескриптор передачи (Transfer Descriptor, сокращенно — TD) описывает параметры транзакции, запрашиваемой клиентом USB. Начало дескриптора должно быть выровнено на границу в 16 байт.

Поле	Смещение
Указатель на следующий элемент списка	00-03h
Слово управления и состояния	04-07h
Маркер дескриптора	08-0Bh
Указатель на буфер данных	0C-0Fh
Зарезервировано для использования программным обеспечением	10-13h 14-17h 18-1Bh 1C-1Fh

Рис. 8.4. Структура дескриптора передачи

Несмотря на то, что USB может выполнять передачи четырех различных типов, все дескрипторы имеют одинаковую структуру, изображенную на рис. 8.4. Каждый дескриптор передачи занимает 32 байта памяти и состоит из двух частей: младшие четыре 32-разрядных слова занимает область данных хост-контроллера, старшие четыре слова — область данных программного обеспечения:

- двойное слово 0 (байты 00-03h) — указатель на следующий элемент списка дескрипторов;
- двойное слово 1 (байты 04-07h) — слово управления и состояния дескриптора передачи;
- двойное слово 2 (байты 08-0Bh) — маркер дескриптора передачи;
- двойное слово 4 (байты 0C-0Fh) — указатель на буфер данных;
- слова 5–8 зарезервированы для использования программным обеспечением.

Область данных программного обеспечения хост-контроллером не обрабатывается и на его функционирование никак не влияет.

31–4	3	2	1	0
Указатель на следующий элемент списка	0	Vf	Q	T

Рис. 8.5. Структура указателя на следующий элемент списка дескрипторов

Указатель на следующий элемент списка дескрипторов имеет структуру, изображенную на рис. 8.5:

- бит 0 (T) — признак последнего элемента списка (0 — указатель содержит адрес следующего элемента списка, 1 — данный элемент является последним в списке, и поле указателя не должно обрабатываться контроллером);
- бит 1 (Q) — тип структуры данных, адрес которой содержится в указателе (0 — дескриптор передачи, 1 — заголовок очереди);
- бит 2 (Vf) — порядок обработки очередей (0 — в ширину, 1 — в глубину);
- бит 3 зарезервирован и должен содержать значение 0;
- биты 4–31 (LP) — биты 4–31 указателя на следующий элемент списка дескрипторов (младшие четыре разряда указателя содержат нули).

Следует уделить особое внимание биту 2, задающему порядок обработки очередей дескрипторов: если этот бит имеет значение 0, то после завершения обработки данного дескриптора контроллер переключится на следующую очередь (обработка списка в ширину), а если бит 2 установлен в 1 — будет обрабатываться следующая транзакция в текущей очереди (обработка списка в глубину).

31 30	29	28 27	28	25	24	23 16	15 11	10 0
Зарез.	SPD	C_Err	LS	ISO	IOC	Status	Зарез.	ActLen

Рис. 8.6. Структура слова управления и состояния дескриптора передачи

Структура слова управления и состояния дескриптора передачи показана на рис. 8.6. Разряды слова имеют следующее назначение:

- биты 0–10 (ActLen) — объем данных, переданный в результате транзакции (значение данного поля на единицу меньше количества переданных байтов);
- биты 11–15 зарезервированы и должны содержать нули;
- бит 16 зарезервирован и должен иметь значение 0;
- бит 17 — признак обнаружения ошибки в принимаемой последовательности битов (данный разряд устанавливается в 1, если в принятой последовательности подряд следует более шести единиц);
- бит 18 — признак обнаружения ошибки тайм-аута или ошибки CRC (устанавливается в 1, если устройство не отвечает на запрос

или при выполнении транзакции обнаружено несовпадение контрольной суммы);

- бит 19 — признак отказа от транзакции (устанавливается в 1, если контроллер получил сигнал «NAK» при выполнении транзакции);
- бит 20 — признак обнаружения перекрестных помех (устанавливается в 1, если контроллер зафиксировал возникновение перекрестных помех при выполнении транзакции);
- бит 21 — признак ошибки в буфере данных (устанавливается в 1 при переполнении буфера в процессе приема или опустошении буфера в процессе передачи данных);
- бит 22 — признак сбоя при выполнении транзакции (устанавливается в 1 при обнаружении серьезной ошибки в процессе выполнения транзакции; при установке этого бита контроллер одновременно сбрасывает бит 23);
- бит 23 — признак активного дескриптора (устанавливается в 1 программным обеспечением при включении дескриптора в очередь и сбрасывается хост-контроллером после завершения связанной с данным дескриптором транзакции или после обнаружения фатальной ошибки при ее выполнении);
- бит 24 (IOC) — управление сигналом прерывания, генерируемым по завершении кадра, в котором выполнялась обработка данного дескриптора (0 — прерывание не требуется, 1 — подать сигнал прерывания);
- бит 25 (IOS) — признак дескриптора изохронной передачи (устанавливается в 1 для дескриптора изохронной передачи и в 0 для дескрипторов других типов);
- бит 26 (LS) — тип устройства (имеет значение 0, если целевое устройство является полноскоростным, и 1 — если устройство низкоскоростное);
- биты 27–28 (C_Err) — счетчик ошибок (00b — нет лимита ошибок, 01b — допускается одна ошибка, 10b — допускаются две ошибка, 11b — допускаются три ошибки). Счетчик ошибок работает на вычитание: его значение уменьшается на единицу после каждой неудачной попытки выполнения транзакции (сигнал «NAK» ошибкой не является и на значение счетчика не влияет). После исчерпания лимита ошибок транзакция становится неактивной, и устанавливается признак сбоя (бит 22);

- бит 29 (SPD) — разрешение приема укороченного пакета данных (0 — прием запрещен, 1 — разрешен). Если данный разряд установлен и длина принимаемого пакета меньше заданной, дескриптор передачи становится неактивным, заголовок очереди не изменяется и (по окончании кадра) устанавливается бит USBINT в регистре состояния и вырабатывается прерывание (если оно разрешено);
- биты 30–31 зарезервированы и должны содержать нули.

Биты 16–23 совместно образуют поле состояния процесса выполнения команды (Status Field), которое перезаписывается хост-контроллером после завершения транзакции. Кроме того, после транзакции хост может записывать информацию в поле ActLen и декрементировать счетчик ошибок. Остальные поля заполняются программным обеспечением в процессе создания дескриптора и в дальнейшем не изменяются.

31 … 21	20	19	18 … 15	14 … 8	7 … 0
MaxLen	R	D	EndPt	Адрес устройства	PID

Рис. 8.7. Структура маркера дескриптора передачи

Маркер дескриптора передачи (TD Token) содержит заголовок пакета, который передается в стартовом маркере USB. Структура маркера показана на рис. 8.7. Разряды слова маркера имеют следующее назначение:

- биты 0–7 (PID) — идентификатор пакета (2Dh — SETUP, 69h — IN, E1h — OUT);
- биты 8–14 — адрес устройства;
- биты 15–18 (EndPt) — номер конечной точки;
- бит 19 (D) — переключатель синхронизации данных (0 — DATA0, 1 — DATA1);
- бит 20 зарезервирован и должен содержать значение 0;
- биты 21–31 (MaxLen) — объем передаваемых данных в байтах минус единица (000h — 1 байт, ..., 4FFh — 1280 байт; 7FFh - пустой пакет (пакет нулевой длины); значения 500h–7FEh считаются недопустимыми).

Указатель на буфер данных содержит 32-разрядный адрес буфера, предназначенного для приема или передачи данных. Объем буфера должен быть не меньше, чем максимальный объем передаваемых

данных, закодированный в поле MaxLen маркера дескриптора передачи.

Заголовок очереди

Заголовок очереди (Queue Head, сокращенно QH) — это специальная структура данных, предназначенная для создания очередей, используемых при передачах типов Control, Bulk и Interrupt.

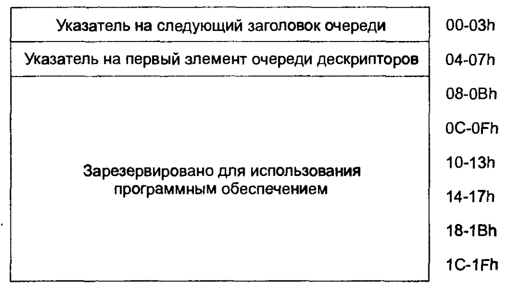


Рис. 8.8. Структура заголовка очереди

Структура заголовка очереди показана на рис. 8.8. Заголовок очереди состоит из двух 32-разрядных слов:

- двойное слово 0 (байты 00-03h) — указатель на следующий элемент «горизонтального» списка;
- двойное слово 1 (байты 04-08h) — указатель на первый элемент очереди.

Заголовок очереди должен быть выровнен на границу 16 байт.

Указатель на следующий заголовок очереди имеет структуру, изображенную на рис. 8.9:

- бит 0 (T) — признак последнего заголовка в списке (0 — указатель содержит адрес следующего заголовка, 1 — данный элемент является последним в «горизонтальном» списке и поле указателя не должно обрабатываться контроллером);
- бит 1 (Q) — тип структуры данных, адрес которой содержится в указателе (0 — дескриптор передачи, 1 — заголовок очереди);
- биты 2–3 зарезервированы и должны содержать нули;

- биты 4–31 (QHLP) — биты 4–31 указателя на следующий элемент «горизонтального» списка (младшие четыре разряда указателя содержат нули).

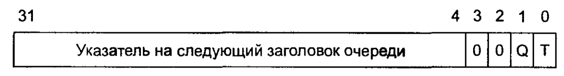


Рис. 8.9. Структура указателя на следующий заголовок очереди

Указатель на первый элемент очереди дескрипторов передачи имеет структуру, изображенную на рис. 8.10:

- бит 0 (T) – признак последнего элемента очереди (0 — указатель содержит адрес следующего элемента, 1 — данный элемент является последним в очереди, и поле указателя не должно обрабатываться контроллером);
- бит 1 (Q) – тип структуры данных, адрес которой содержится в указателе (0 — дескриптор передачи, 1 — заголовок очереди);
- биты 2–3 зарезервированы и должны содержать нули;
- биты 4–31 (QHLP) – биты 4–31 указателя на следующий элемент очереди (младшие четыре разряда указателя содержат нули).

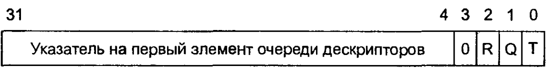


Рис. 8.10. Структура указателя на первый элемент очереди дескрипторов передачи

Порядок обработки списка дескрипторов

Порядок выполнения запросов определяется структурой списка дескрипторов, который в спецификации хост-контроллера [91] именуется «планом» (Schedule). Для обеспечения нормальной работы шины USB дескрипторы в списке должны размещаться в определенном порядке, как показано на рис. 8.11.

ПРИМЕЧАНИЕ

Каждый элемент списка кадров может быть действительным или недействительным Действительный элемент списка кадров должен содержать указатель на дескриптор передачи или очередь заголовка. В нормальном режиме работы контроллера все элементы списка должны быть действительными.

Когда начинается выполнение очередного кадра, хост-контроллер получает из списка кадров указатель на список дескрипторов, который должен быть обработан в данном кадре.

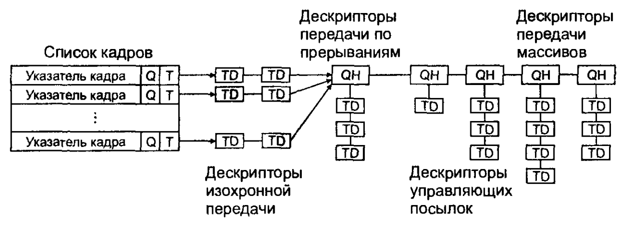


Рис. 8.11. Пример построения списка дескрипторов

В первую очередь должны быть обработаны дескрипторы изохронной передачи, поэтому такие дескрипторы размещаются последовательно друг за другом в начале списка. Далее обрабатывается список заголовков, в начале которого размещены заголовки очередей дескрипторов для передачи по прерываниям, затем следуют заголовки очередей дескрипторов управляющих посылок. В конце списка находятся заголовки очередей дескрипторов передачи массивов данных.

ПРИМЕЧАНИЕ

Список дескрипторов изохронной передачи в каждом кадре свой, а список заголовков очередей — общий для всех кадров.

Каждый заголовок очереди указывает на первый из находящихся в очереди дескрипторов передачи. Обычно список заголовков очередей обрабатывается по горизонтали (в ширину): контроллер извлекает из заголовка первый дескриптор в очереди, обрабатывает его и переходит к следующему заголовку очереди. При необходимости можно установить режим обработки очереди по вертикали: контроллер в этом случае вначале обработает всю очередь и только потом перейдет к заголовку следующей очереди.

Если выполнение операции, заданной дескриптором передачи, не завершено в текущем кадре (например, данные не готовы для передачи), в следующем кадре операция повторяется. Когда операция завершена, дескриптор передачи помечается как обслуженный и удаляется из очереди: контроллер извлекает из него указатель на следующий дескриптор и переписывает его в заголовок очереди. Область,

зарезервированная в дескрипторе передачи для программного обеспечения, в первую очередь предназначена для «сборки мусора»: из обслуженных и не нужных более дескрипторов можно сформировать очередь с целью повторного использования занимаемых ими участков памяти.

Каждая очередь дескрипторов передачи обычно формируется прикладной программой для работы с определенной функцией или даже одной конечной точкой функции. Если программа работает с несколькими конечными точками или в системе параллельно выполняется несколько прикладных программ, в списке будут присутствовать несколько очередей дескрипторов.

Запросы к устройствам USB

Все устройства USB принимают запросы от хост-контроллера и отвечают на них через Основной канал сообщений. Запросы выполняются при помощи управляющих посылок.

Запрос и его параметры передаются устройству в Setup-пакете, структура которого показана в табл. 8.3. Каждый Setup-пакет имеет размер 8 байт.

Таблица 8.3. Структура Setup-пакета

Смещение	Мнемоника	Размер	Описание
0	bmRequestType	BYTE	Характеристики запроса: биты 0–4 — код получателя (0 — устройство, 1 — интерфейс, 2 — другой получатель; коды 4–31 зарезервированы); биты 5–6 — код типа запроса (0 — стандартный запрос, 1 — специфический запрос для данного класса, 2 — специфический запрос изготовителя, 3 — зарезервирован); бит 7 — направление передачи данных (0 — от хоста к устройству, 1 — от устройства к хосту)
1	bRequest	BYTE	Код запроса
2	wValue	WORD	Параметр запроса
4	wIndex	WORD	Индекс или смещение
6	wLength	WORD	Количество байтов, подлежащих передаче на стадии передачи данных

Поле кода запроса определяет тип запроса. В спецификации USB определены только коды стандартных запросов к устройству:

- 0 — GET_STATUS (определить состояние устройства);
- 1 — CLEAR_FEATURE (сбросить свойство);
- 3 — SET_FEATURE (установить свойство);
- 5 — SET_ADDRESS (установить адрес);
- 6 — GET_DESCRIPTOR (получить дескриптор);
- 7 — SET_DESCRIPTOR (загрузить дескриптор);
- 8 — GET_CONFIGURATION (получить код текущей конфигурации);
- 9 — SET_CONFIGURATIDN (установить конфигурацию);
- 10 — GET_INTERFACE (получить код интерфейса);
- 11 — SET_INTERFACE (установить интерфейс);
- 12 — SYNCH_FRAME (кадр синхронизации).

Значение параметров wValue, wIndex зависят от типа запроса. В запросах на прием или передачу дескрипторов параметр wValue содержит тип дескриптора в старшем байте и индекс дескриптора — в младшем. Каждому типу дескрипторов поставлен в соответствие определенный числовой код:

- 1 (DEVICE) — дескриптор устройства;
- 2 (CONFIGURATION) — дескриптор конфигурации;
- 3 (STRING) — дескриптор строки;
- 4 (INTERFACE) — дескриптор интерфейса;
- 5 (ENDPOINT) — дескриптор конечной точки.

Поле wIndex обычно используется для задания номера интерфейса или конечной точки. Если поле wIndex задает конечную точку, оно имеет следующий формат:

- биты 0–3 — номер конечной точки;
- биты 4–6 зарезервированы и должны содержать нули;
- бит 7 — направление передачи конечной точки (0 — OUT, 1 — IN);
- биты 8–15 зарезервированы и должны содержать нули.

Если поле wIndex задает номер интерфейса, то младший байт (биты 0–7) содержит номер интерфейса, а старший байт не используется (биты 8–15 зарезервированы и должны содержать нули).

Запрос Get Status позволяет определить состояние устройства, интерфейса или конечной точки. Запрос имеет следующие параметры:

- поле bmRequestType уточняет запрос (10000000b – получить состояние устройства, 10000001b – получить состояние интерфейса, 10000010b – получить состояние конечной точки);
- wValue = 0;
- wIndex – ноль (если запрос обращен к устройству), номер интерфейса или конечной точки;
- wLength = 2.

По запросу Get Status устройство возвращает 16-разрядное слово состояния, описывающее текущее состояние устройства, интерфейса или конечной точки.

Разряды слова состояния устройства имеют следующее назначение:

- бит 0 (Self Powered) – режим электропитания (0 – устройство получает питание от шины USB, 1 – от собственного источника энергии);
- бит 1 (Remote Wakeup) – реакция на сигнал пробуждения от шины USB (0 – устройство игнорирует сигнал, 1 – устройство реагирует на сигнал);
- биты 2–15 зарезервированы и должны содержать нули.

Слово состояния интерфейса зарезервировано и содержит нули во всех разрядах.

Разряды слова состояния конечной точки имеют следующее назначение:

- бит 0 (Halt) – признак «зависания» конечной точки (0 – конечная точка функционирует нормально, 1 – передача данных заблокирована);
- биты 1–15 зарезервированы и должны содержать нули.

Запрос Clear Feature используется для того, чтобы запретить свойство или состояние, указанное значением селектора свойств. Запрос имеет следующие параметры:

- поле bmRequestType уточняет запрос (00000000b – запретить свойство устройства, 00000001b – запретить свойство интерфейса, 00000010b – запретить свойство конечной точки);
- wValue – селектор свойств;
- wIndex – ноль (если запрос обращен к устройству), номер интерфейса или конечной точки;
- wLength = 0.

В спецификации USB 1.1 определены только два значения селектора свойств:

- 0 (обозначение — ENDPOINT_HALT, получатель — конечная точка) — блокировка конечной точки;
- 1 (обозначение — DEVICE_REMOTE_WAKEUP, получатель — устройство) — разрешить выполнение сигнала пробуждения.

Передача данных по запросу Clear Feature не производится. Сброс состояния ENDPOINT_HALT разблокирует конечную точку; сброс состояния DEVICE_REMOTE_WAKEUP лишает устройство способности реагировать на сигнал пробуждения.

Запрос Set Feature используется для того, чтобы разрешить свойство или состояние, указанное значением селектора свойств. Запрос имеет следующие параметры:

- поле bmRequestType уточняет запрос (00000000b — разрешить свойство устройства, 00000001b — разрешить свойство интерфейса, 00000010b — разрешить свойство конечной точки);
- wValue — селектор свойства;
- wIndex — ноль (если запрос обращен к устройству), номер интерфейса или конечной точки;
- wLength = 0.

Передача данных по запросу Set Feature не производится. Установка состояния ENDPOINT_HALT блокирует конечную точку; установка состояния DEVICE_REMOTE_WAKEUP позволяет устройству реагировать на сигнал пробуждения.

Запрос Set Address позволяет присвоить устройству новое значение адреса на шине USB. Запрос имеет следующие параметры:

- bmRequestType = 00000000b;
- wValue — адрес устройства;
- wIndex = 0;
- wLength = 0.

Передача данных при выполнении запроса Set Address не производится.

Запрос Get Descriptor позволяет получить дескриптор устройства, дескриптор конфигурации или дескриптор строки. Запрос имеет следующие параметры:

- bmRequestType = 10000000b;
- wValue содержит тип дескриптора в старшем байте (DEVICE — дескриптор устройства, CONFIGURATION — дескриптор конфигурации,

STRING — дескриптор строки) и индекс дескриптора в младшем байте (при запросе дескриптора устройства индекс имеет значение 0);

- wIndex — ноль (для дескриптора устройства или конфигурации) или идентификатор языка (для дескриптора строки);
- wLength — размер дескриптора в байтах.

ПРИМЕЧАНИЕ

По запросу на получение дескриптора конфигурации устройство выдает дескриптор конфигурации, дескрипторы интерфейсов, а также дескрипторы конечных точек для всех интерфейсов. Первый дескриптор интерфейса следует за дескриптором конфигурации, а дескрипторы конечных точек следуют за дескриптором интерфейса Если устройство имеет дополнительные интерфейсы, то после дескриптора последней конечной точки первого интерфейса будет передаваться дескриптор второго интерфейса, дескрипторы конечных точек второго интерфейса и т. д. Специфические дескрипторы класса и дескрипторы изготовителя следуют за теми стандартными дескрипторами, значение которых они дополняют или модифицируют.

По запросу Get Descriptor хаб передает хосту дескриптор, тип которого указан в запросе.

Запрос Set Descriptor позволяет дополнить существующий (или добавить новый) дескриптор устройства, конфигурации или строки. Запрос имеет следующие параметры:

- bmRequestType = 00000000b;
- wValue — тип дескриптора и индекс дескриптора;
- wIndex — ноль или идентификатор языка;
- wLength — размер дескриптора в байтах.

В процессе выполнения запроса Set Descriptor хост передает периферийному устройству дескриптор, тип которого определяется параметрами запроса.

По запросу Get Configuration устройство выдает код своей текущей конфигурации. Запрос имеет следующие параметры:

- bmRequestType = 10000000b;
- wValue = 0;
- wIndex = 0;
- wLength = 1.

При выполнении запроса Get Configuration от устройства к хосту передается один байт данных, содержащий код конфигурации устройства.

Запрос Set Configuration позволяет задать устройству новую конфигурацию. Запрос имеет следующие параметры:

- bmRequestType = 00000000b;
- wValue — код конфигурации;
- wIndex = 0;
- wLength = 1;
- передача данных не производится.

При выполнении запроса Set Configuration от хоста к устройству передается один байт данных, содержащий код конфигурации, которую требуется установить.

Запрос Get Interface позволяет получить код текущей настройки для указанного интерфейса. Запрос имеет следующие параметры:

- bmRequestType = 10000001b;
- wValue = 0;
- wIndex — номер интерфейса;
- wLength = 1.

При выполнении запроса Get Interface от устройства к хосту передается один байт данных, содержащий код текущего варианта настройки интерфейса.

Запрос Set Interface позволяет задать новый вариант настройки для указанного интерфейса. Запрос имеет следующие параметры:

- bmRequestType = 00000001b;
- wValue — код варианта настройки интерфейса;
- wIndex — номер интерфейса;
- wLength = 0.

Передача данных при выполнении запроса Set Interface не производится.

Запрос Synch Frame используется для задания номера кадра синхронизации. Запрос имеет следующие параметры:

- bmRequestType = 10000010b;
- wValue = 0;
- wIndex — номер конечной точки;
- wLength = 2.

С помощью запроса Set Interface хост передает заданной конечной точке, работающей в изохронном режиме, 16-разрядное слово данных, содержащее номер кадра, который конечная точка должна использовать для синхронизации передачи.

Стандартные дескрипторы USB

В спецификации на шину USB указана группа дескрипторов, которые должны выдаваться устройствами USB в ответ на стандартные запросы. Структура таких дескрипторов стандартизирована, а в документации они именуются стандартными дескрипторами (standard descriptors).

Дескриптор устройства

Стандартный дескриптор устройства (Standard Device Descriptor) содержит основную информацию об устройстве USB. Структура Стандартного дескриптора устройства показана в табл. 8.4.

Таблица 8.4. Структура Стандартного дескриптора устройства

Смещение	Мнемоника	Размер	Описание
0	bLength	BYTE	Размер данного дескриптора в байтах
1	bDescriptorType	BYTE	Тип дескриптора (DEVICE)
2	bcdUSB	WORD	Номер версии спецификации USB, которой соответствует дескриптор, представленный в двоично-десятичном формате BCD
4	bDeviceClass	BYTE	Код класса устройства USB (если в данном поле записано значение 0, то интерфейсы функционируют независимо друг от друга и каждый из них имеет собственный код класса; если в поле записано значение FFh, то класс устройства определяется изготовителем)
5	bDeviceSubClass	BYTE	Код подкласса устройства USB
6	bDeviceProtocol	BYTE	Код протокола USB (если в данном поле записано значение 0, то устройство не использует специфические протоколы своего класса, однако может использовать специфическое протоколы интерфейса; если в поле записано значение FFh, то устройство использует протокол, определяемый изготовителем)
7	bMaxPacketSize0	BYTE	Максимальный размер пакета для нулевой конечной точки (допускается использование значений 8, 16, 32 и 64)
8	idVendor	WORD	Идентификатор изготовителя устройства

продолжение ⇗

Таблица 8.4 (*продолжение*)

Смещение	Мнемоника	Размер	Описание
10	idProduct	WORD	Идентификатор продукта (определяется изготовителем)
12	bcdDevice	WORD	Номер версии устройства, представленный в двоично-десятичном формате BCD
14	iManufacturer	BYTE	Индекс дескриптора строки, описывающей изготовителя
15	iProduct	BYTE	Индекс дескриптора строки, описывающей продукт
16	iSerialNumber	BYTE	Индекс дескриптора строки, содержащей серийный номер устройства
17	bNumConfigurations	BYTE	Количество возможных конфигураций устройства

Некоторые поля Стандартного дескриптора устройства традиционно содержат фиксированные значения и не несут информационной нагрузки:

- размер стандартного дескриптора всегда составляет 18 байт;
- код типа дескриптора имеет значение 1;
- код подкласса имеет значение 0;
- код протокола имеет значение 0;
- максимальный размер пакета для Основного канала сообщений всегда составляет 8 байт.

Код версии спецификации USB может принимать следующие значения: 0100h — версия 1.0, 0110h — версия 1.1, 0200h — версия 2.0.

Значение кода класса равно 9 для хабов и 0 для любых других устройств, поэтому определить тип устройства при помощи дескриптора устройства можно только в том случае, если оно является хабом, а идентификация других стандартных периферийных устройств выполняется по дескриптору интерфейса.

Идентификатор изготовителя устройства, идентификатор продукта и номер версии устройства обычно представляют интерес только для разработчиков драйверов, которые работают в фирме, выпускающей данный продукт: по ним можно однозначно определить тип и марку устройства, но для этого нужно иметь доступ к фирменной документации.

Индексы дескрипторов строк используются для получения информации об устройстве в текстовом формате: при подаче запроса на получение дескриптора строки индекс дескриптора передается в младшем байте параметра wValue.

Значение последнего байта Стандартного дескриптора устройства показывает, сколько различных вариантов конфигурации можно задать для данного устройства. Как правило, периферийные устройства, предназначенные для использования совместно с персональными компьютерами, имеют только один вариант конфигурации.

Дескриптор конфигурации

Стандартный дескриптор конфигурации (Standard Configuration Descriptor) содержит информацию об одной из возможных конфигураций устройства. Структура Стандартного дескриптора конфигурации показана в табл. 8.5.

Таблица 8.5. Структура Стандартного дескриптора конфигурации

Смещение	Мнемоника	Размер	Описание
0	bLength	BYTE	Размер данного дескриптора в байтах
1	bDescriptorType	BYTE	Тип дескриптора (CONFIGURATION)
2	wTotalLength	WORD	Общий объем данных в байтах, возвращаемый для этой конфигурации (суммарная длина всех дескрипторов для этой конфигурации)
4	bNumInterfaces	BYTE	Количество интерфейсов, поддерживаемых данной конфигурацией
5	bConfigurationValue	BYTE	Значение, которое должно использоваться в качестве аргумента в запросе SET_CONFIGURATION для установки данной конфигурации
6	iConfiguration	BYTE	Индекс дескриптора строки, описывающей данную конфигурацию
7	bmAttributes	BYTE	Характеристики конфигурации: биты 0–4 зарезервированы и должны содержать нули; бит 5 — признак возможности пробуждения устройства по внешнему сигналу (устанавливается в единицу, если данная возможность поддерживается);

продолжение ⇨

Таблица 8.5 (*продолжение*)

Смещение	Мнемоника	Размер	Описание
			бит 6 — признак наличия собственного источника питания (0 — устройство получает питание по шине USB, 1 — имеет собственный источник питания); бит 7 зарезервирован и должен быть установлен в 1
8	MaxPower	BYTE	Код мощности, потребляемой устройством от шины USB (максимальный ток в миллиамперах, потребляемый устройством от шины, вычисляется путем умножения значения данного поля на два)

Устройство может иметь один или несколько дескрипторов конфигурации в соответствии с количеством возможных конфигураций, указанных в Стандартном дескрипторе устройства.

Каждая конфигурация описывается одним стандартным дескриптором. Размер Стандартного дескриптора конфигурации всегда составляет 9 байт, а код типа дескриптора имеет значение 2.

ВНИМАНИЕ

По запросу «Получить Стандартный дескриптор конфигурации» устройство выдает не только дескриптор конфигурации, но и все имеющиеся дескрипторы интерфейсов и конечных точек.

Для определения общей длины (в байтах) возвращаемого списка дескрипторов служит поле `wTotalLength`. Чтобы получить весь список дескрипторов, нужно запросить от устройства первые 8 байт Стандартного дескриптора конфигурации, запомнить значение поля `wTotalLength`, а затем использовать это значение в качестве параметра при повторной подаче запроса.

Каждая конфигурация может иметь один или несколько интерфейсов. Количество доступных интерфейсов указывается в поле `bNumInterfaces`.

Дескриптор интерфейса

Стандартный дескриптор интерфейса (Standard Interface Descriptor) содержит информацию об одном из интерфейсов, доступных

при определенной конфигурации устройства. Структура Стандартного дескриптора интерфейса показана в табл. 8.6.

Таблица 8.6. Структура Стандартного дескриптора интерфейса

Смещение	Мнемонике	Размер	Описание
0	bLength	BYTE	Размер данного дескриптора в байтах
1	bDescriptor Type	BYTE	Тип дескриптора (INTERFACE)
2	bInterface-Number	BYTE	Порядковый номер интерфейса в данной конфигурации
3	bAlternateSetting	BYTE	Код варианта для данного интерфейса
4	bNumEndpoints	BYTE	Количество конечных точек, используемых данным интерфейсом, за вычетом Нулевой конечной точки. Если значение этого поля равно нулю, интерфейс может использовать только основной канал сообщений
5	bInterfaceClass	BYTE	Код класса интерфейса (нулевое значение кода зарезервировано; код FFh указывает, что интерфейс определяется изготовителем)
6	bInterfaceSub-Class	BYTE	Код подкласса интерфейса
7	bInterface-Protocol	BYTE	Код протокола (если поле имеет значение FFh, то протокол определяется изготовителем)
8	iInterface	BYTE	Индекс дескриптора строки, описывающей интерфейс

Размер Стандартного дескриптора интерфейса всегда составляет 9 байт, а код типа дескриптора имеет значение 4.

ПРИМЕЧАНИЕ

При выполнении идентификации и нумерации устройств на шине USB дескриптор интерфейса может использоваться для определения типа устройства по кодам класса, подкласса и протокола.

Дескриптор конечной точки

Стандартный дескриптор конечной точки (Standard Endpoint Descriptor) содержит информацию об одной из конечных точек,

доступных при использовании определенного интерфейса. Структура Стандартного дескриптора конечной точки показана в табл. 8.7.

Таблица 8.7. Структура Стандартного дескриптора конечной точки

Смещение	Мнемоника	Размер	Описание
0	bLength	BYTE	Размер данного дескриптора в байтах
1	bDescriptorType	BYTE	Тип дескриптора (INTERFACE)
2	bEndpoint-Address	BYTE	Код адреса конечной точки, описываемой данным дескриптором: биты 0–3 — номер конечной точки; биты 4–6 зарезервированы и должны содержать нули; бит 7 — направление передачи (0 — OUT, 1 — IN). Для канала сообщений направление игнорируется
3	bmAttributes	BYTE	Атрибуты конечной точки. Используются биты 0 и 1 (остальные разряды зарезервированы и содержат нули): 00b — канал сообщений, 01b — изохронный канал, 10b — канал сплошной передачи; 11b — канал прерываний
4	wMaxPacket-Size	WORD	Максимальный размер пакета для конечной точки
6	bInterval	BYTE	Интервал опроса конечной точки при передаче данных (задается в миллисекундах)

Размер Стандартного дескриптора интерфейса всегда составляет 7 байт, а код типа дескриптора имеет значение 5.

Код адреса `bEndpointAddress` и байт атрибутов `bmAttributes` для многих классов периферийных устройств позволяют однозначно определить функциональное назначение конечной точки.

Поле размера пакета `wMaxPacketSize` задает предельный размер пакета данных для конечной точки.

Интервал опроса конечной точки имеет значение только в том случае, если точка используется для передачи данных по прерываниям: если точка является точкой сообщений или сплошной передачи, значение поля `bInterval` игнорируется, а для изохронных конечных точек поле `bInterval` всегда содержит значение 1.

Дескриптор строки

Дескриптор строки (UNICODE String Descriptor) содержит текст в формате UNICODE. Строка не ограничивается нулем: длина строки вычисляется путем вычитания 2 из размера дескриптора. Структура Дескриптора строки показана в табл. 8.8.

Таблица 8.8. Структура Дескриптора строки

Смещение	Мнемоника	Размер	Описание
0	bLength	BYTE	Размер данного дескриптора в байтах (N + 2)
1	bDescriptorType	BYTE	Тип дескриптора (STRING)
2	bString	N байт	Строка символов UNICODE

Дескриптор строки не является обязательным. Если устройство не поддерживает дескрипторы строк, все ссылки на такие дескрипторы из дескрипторов устройства, конфигурации или интерфейса должны иметь нулевое значение.

Устройство может поддерживать несколько различных языков, поэтому при запросе дескриптора строки нужно задавать 16-разрядный идентификатор языка (LANGID).

Строковый индекс 0 для всех языков соответствует дескриптору строки, содержащей массив 16-разрядных идентификаторов всех поддерживаемых устройством языков. Массив идентификаторов не ограничен нулем: размер массива в байтах вычисляется путем вычитания 2 из размера дескриптора. Структура массива следующая: байт 0 содержит размер дескриптора в байтах (число идентификаторов + 2), байт 1 описывает тип дескриптора (STRING), далее следуют 16-разрядные идентификаторы языка.

Примеры использования стандартных дескрипторов приведены в листингах 8.1 и 8.2. Листинг 8.1 содержит набор процедур общего назначения, предназначенных для работы с устройствами USB:

- процедура FindUSBController предназначена для выполнения последовательного поиска хост-контроллеров USB на шине PCI при помощи функций PCI BIOS (в случае обнаружения очередного контроллера процедура запоминает его параметры и возвращает управление вызывающей программе);
- процедура Wait05s обеспечивает задержку выполнения программы на один тик системного таймера;

- процедура InitializeDescriptors формирует основные структуры данных хост-контроллера USB;
- процедура StatusIN_Transaction предназначена для считывания дескрипторов состояния устройства;
- процедура Setup_Transaction предназначена для установки параметров устройства (в первую очередь — номера устройства на шине USB);
- процедура Enumeratin использует процедуру Setup_Transaction для присвоения устройству порядкового номера;
- процедура GetConfigurationDescriptor использует процедуру StatusIN_Transaction для получения от устройства стандартного дескриптора конфигурации.

Листинг 8.1. Процедуры для работы с устройствами USB

```
DATASEG
; Признак успешного завершения поиска
SearchResult  DB 0
; Индекс хост-контроллера
USB_HostIndex DW 0
; ПАРАМЕТРЫ КОНТРОЛЛЕРА USB
; Координаты устройства PCI
USB_BusNum    DB ? ;номер шины
USB_DevNum    DB ? ;номер устройства и номер функции
; Идентификаторы устройства PCI
USB_VendorID  DW ? ;идентификатор изготовителя
USB_DeviceID  DW ? ;идентификатор устройства
; Адрес блока регистров контроллера PCI USB
USB_BaseAddr  DW ?
; Номер используемого прерывания IRQ
USB_IntLine   DB ?
; Сообщения об ошибках
BadRg DB LIGHTRED,12,28,"Неверный номер регистра",0
NoPCI DB 12,24,"Система не поддерживает PCI BIOS",0
NoUSB DB 12,28,"Контроллер USB не найден",0
NoDev DB 12,26,"Устройство USB не обнаружено",0
TmOut DB 12,21,"Превышен допустимый интервал ожидания",0
BfErr DB 12,28,"Буфер данных переполнен",0
DsErr DB 12,25,"Список дескрипторов переполнен",0
; ПАРАМЕТРЫ УСТРОЙСТВА
; Номер используемого устройством порта USB
USB_PortNum DB ?
; Адрес регистра состояния используемого порта
USB_PortReg DW 0
; Тип устройства:
; 0 - полноскоростное, 1 - низкоскоростное
USB_Device_Type   DB 0
```

```
; Номер устройства на шине USB
USB_Device_Nuшber DB 0
; Номер конечной точки
USB_EnpointNumber DB 0
; ПРЕОБРАЗОВАННЫЕ ЗНАЧЕНИЯ ПАРАМЕТРОВ УСТРОЙСТВА
; Сдвинутый влево на 7 разрядов номер функции
ShFuncNum  DD ?
; Сдвинутый влево на 15 разрядов номер конечной точки
ShEndpNum  DD ?
; Сдвинутый влево на 26 разрядов тип устройства
ShDevType  DD ?
; Сдвинутый влево на 21 размер пакета
ShPackSize DD ?
; ФИЗИЧЕСКИЕ АДРЕСА СТРУКТУР ДАННЫХ USB
; Линейный адрес заголовка списка дескрипторов
Addr_QH           DD ?
; Линейный адрес массива дескрипторов передачи
Addr_TD_Array     DD ?
; Линейный адрес дескриптора команды
Addr_CommandDescr DD ?
; Линейный адрес буфера данных
Addr_DataDescr    DD ?
; ПАРАМЕТРЫ ТРАНЗАКЦИЙ
; Номер кадра в момент начала транзакции
StartFrame    DW ?
; Состояние триггера данных
DataTrigger   DD ?
; Объем данных в передаваемом массиве
BULK_DataSize DW 0
; БУФЕР ДАННЫХ
DataBuffer DB 4096 DUP(?)
ENDS

CODESEG
;*****************************************************
;*      НАЙТИ ХОСТ-КОНТРОЛЛЕР USB И ОПРЕДЕЛИТЬ       *
;*               ЕГО ОСНОВНЫЕ ПАРАМЕТРЫ               *
;* Подпрограмма выполняет поиск хост-контроллеров    *
;* USB, подключенных к шине PCI.                      *
;* Входные параметры:                                 *
;* USB_HostIndex - индекс хост-контроллера.           *
;* В случае успешного завершения операции поиска     *
;* параметры контроллера сохраняются в глобальных    *
;* переменных USB_BusNum, USB_DevNum, USB_VendorID,  *
;* USB_DeviceID, USB_BaseAddr и USB_IntLine.         *
;*****************************************************
PROC FindUSBController near
        pushad
; Найти первый USB-контроллер по коду класса
        mov     AX,0B103h
```

продолжение

Листинг 8.1 (*продолжение*)

```
        mov     ECX,0C0300h
        mov     SI,[USB_HostIndex]
        int     1Ah
        jnc     @@ReadPCIRegisters ;устройство найдено
; Выход: контроллер USB не найден
        mov     [SearchResult],1
        popad
        ret
; Устройство обнаружено, его координаты на шине PCI
; находятся в регистре BX
@@ReadPCIRegisters:
; Запомнить координаты контроллера
        mov     [USB_BusNum],BH
        mov     [USB_DevNum],BL
; Получить идентификатор изготовителя
        mov     AX,0B109h ;читать слово
        mov     DI,0      ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [USB_VendorID],CX
; Получить идентификатор устройства
        mov     AX,0B109h ;читать слово
        mov     DI,2      ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [USB_DeviceID],CX
; Получить базовый адрес блока регистров контроллера USB
        mov     AX,0B10Ah ;читать двойное слово
        mov     DI,20h    ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        ; Обнулить 5 младших бит
        and     CX,0FFE0h
        ; Сохранить только младшее слово адреса
        mov     [USB_BaseAddr],CX
; Получить номер используемого устройством
; прерывания IRQ
        mov     AX,0B108h ;читать байт
        mov     DI,3Ch    ;смещение байта
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [USB_IntLine],CL
; Выход: контроллер найден, увеличить значение индекса
        inc     [USB_HostIndex]
        popad
        ret
; ОБРАБОТКА ОШИБОК
; Неверный номер регистра
@@BadRegisterNumber:
```

```
        MFatalError BadRg
ENDP FindUSBController

;*****************************************
;* ЗАДЕРЖКА НА ОДИН ТИК СИСТЕМНОГО ТАЙМЕРА *
;*****************************************
PROC Wait05s near
        push    ES
        push    EAX
        mov     AX,0
        mov     ES,AX
        mov     EAX,[ES:046Ch]
        inc     EAX
@@Wait: cmp     EAX,[ES:046Ch]
        jae     @@Wait
        pop     EAX
        pop     ES
        ret
ENDP Wait05s

;***********************************
;* ИНИЦИАЛИЗИРОВАТЬ ДЕСКРИПТОРЫ USB *
;***********************************
PROC InitializeDeascriptors near
        pushad
; Очистить 1 Мбайт памяти
        mov     EBX,FrameListBaseAddr
        mov     ECX,100000h/4
        xor     EAX,EAX
@@ClearMemory:
        mov     [GS:EBX],EAX
        add     EBX,4
        dec     ECX
        jnz     @@ClearMemory

; ПОДГОТОВИТЬ СТРУКТУРЫ ДАННЫХ ДЛЯ РАБОТЫ С USB
        ; Вычислить линейный адрес буфера данных
        xor     EAX,EAX
        xor     EBX,EBX
        mov     AX,DS
        shl     EAX,4
        mov     BX,offset DataBuffer
        add     EAX,EBX
        mov     [Addr_DataDescr],EAX
        ; Вычислить линейный адрес сегмента дескрипторов
        xor     EAX,EAX
        mov     AX,USB_DESCR
        shl     EAX,4
        mov     EDX,EAX     ;запомнить адрес сегмента
        ; Вычислить линейный адрес заголовка списка
```

продолжение

Листинг 8.1 (*продолжение*)

```
        xor     EBX,EBX
        mov     BX,offset QH_Descriptor
        add     EAX,EBX
        mov     [Addr_QH],EAX
        ; Вычислить линейный адрес массива дескрипторов
        mov     EAX,EDX
        mov     BX,offset TD_Array
        add     EAX,EBX
        mov     [Addr_TD_Array],EAX
; Настроить указатели кадров на заголовок очереди
        mov     EBX,FrameListBaseAddr
        mov     EAX,[Addr_QH]
        or      EAX,10b              ; T = 0, Q = 1
        mov     CX,1024
@@ActivateNextFrame:
        mov     [GS:EBX],EAX
        add     EBX,4
        loop    @@ActivateNextFrame
        popad
        ret
ENDP InitializeDeascriptors

;*****************************************************
;*        ЗАПРОСИТЬ ИНФОРМАЦИЮ ОБ УСТРОЙСТВЕ USB        *
;* Передаваемые параметры:                              *
;* SI - смещение дескриптора команды в сегменте данных.*
;*****************************************************
PROC StatusIN_Transaction near
        pushad
; Запомнить в DX объем передаваемых данных
        mov     DX,[SI+6]
; Вычислить линейный адрес данных дескриптора команды
        xor     EAX,EAX
        mov     AX,DS
        shl     EAX,4
        and     ESI,0FFFFh
        add     EAX,ESI
        mov     [Addr_CommandDescr],EAX
; Загрузить в ESI указатель на массив дескрипторов
        mov     ESI,[Addr_TD_Array]
; Сформировать дескриптор команды
        ; Указатель на следующий TD
        mov     EAX,ESI
        add     EAX,32
        mov     [GS:ESI],EAX
        ; Слово управления
        mov     EAX,[ShDevType] ;тип устройства
        or      EAX,00800000h   ;признак активности
        mov     [GS:ESI+4],EAX
```

```
        ; Маркер
        mov     EAX,[ShFuncNum] ;номер функции
        or      EAX,00E0002Dh   ;передать 8 байт
        mov     [GS:ESI+8],EAX
        ; Указатель на дескриптор команды
        mov     EAX,[Addr_CommandDescr]
        mov     [GS:ESI+12],EAX
        xor     EAX,EAX
        mov     [GS:ESI+16],EAX
        mov     [GS:ESI+20],EAX
        mov     [GS:ESI+24],EAX
        mov     [GS:ESI+28],EAX
        add     ESI,32

; Вычислить количество 8-байтных блоков
; и размер последнего блока данных
        mov     CX,DX
        shr     CX,3          ;количество 8-байтных блоков
        cmp     CX,64-3
        ja      @@TD_Array_Error
        and     EDX,111b      ;размер последнего блока
        mov     EBX,[Addr_DataDescr]
        mov     EDI,80000h ;триггер данных
        cmp     CX,0
        je      @@ShortDataBlock
@@NextDataBlock:
; Сформировать дескриптор данных
        ; Указатель на следующий TD
        mov     EAX,ESI
        add     EAX,32
        mov     [GS:ESI],EAX
        ; Слово управления
        mov     EAX,[ShDevType] ;тип устройства
        or      EAX,00800000h   ;признак активности
        mov     [GS:ESI+4],EAX
        ; Маркер
        mov     EAX,[ShFuncNum] ;номер функции
        or      EAX,00E00069h   ;принять 8 байт
        or      EAX,EDI         ;триггер данных
        mov     [GS:ESI+8],EAX
        xor     EDI,80000h      ;переключить триггер
        mov     [GS:ESI+12],EBX ;буфер данных
        add     EBX,8
        xor     EAX,EAX
        mov     [GS:ESI+16],EAX
        mov     [GS:ESI+20],EAX
        mov     [GS:ESI+24],EAX
        mov     [GS:ESI+28],EAX
        add     ESI,32
        loop    @@NextDataBlock
```

продолжение ⇗

Листинг 8.1 *(продолжение)*

```
; Имеется короткий блок?
@@ShortDataBlock:
        cmp     DX,0
        je      @@NoShortBlock
; Сформировать дескриптор данных короткого блока
        mov     EAX,ESI
        add     EAX,32
        mov     [GS:ESI],EAX
        ; Слово управления
        mov     EAX,[ShDevType] ;тип устройства
        or      EAX,00800000h   ;признак активности
        mov     [GS:ESI+4],EAX
        ; Маркер
        mov     EAX,[ShFuncNum] ;номер функции
        dec     DX
        shl     EDX,21
        or      EAX,EDX         ;размер блока
        or      EAX,EDI         ;триггер данных
        or      EAX,69h
        mov     [GS:ESI+8],EAX
        mov     [GS:ESI+12],EBX
        xor     EAX,EAX
        mov     [GS:ESI+16],EAX
        mov     [GS:ESI+20],EAX
        mov     [GS:ESI+24],EAX
        mov     [GS:ESI+2B],EAX
        add     ESI,32

@@NoShortBlock:
; Сформировать дескриптор пустого пакета
        mov     [dword ptr GS:ESI],1b ;последний TD
        ; Слово управления
        mov     EAX,[ShDevType] ;тип устройства
        or      EAX,00B00000h   ;признак активности
        mov     [GS:ESI+4],EAX
        ; Маркер
        mov     EAX,[ShFuncNum] ;нонер функции
        or      EAX,0FFEB00E1h  ;передать пустой блок
        mov     [GS:ESI+8],EAX
        xor     EAX,EAX
        mov     [GS:ESI+12],EAX
        mov     [GS:ESI+16],EAX
        mov     [GS:ESI+20],EAX
        mov     [GS:ESI+24],EAX
        mov     [GS:ESI+2B],EAX
; Установить указатель на список дескрипторов
; (контроллер начинает передачу данных)
        mov     EAX,[Addr_TD_Array]
        mov     ESI,[Addr_QH]
```

```
        add     ESI,4
        mov     [GS:ESI],EAX
; Запомнить номер текущего кадра
        mov     DX,[USB_BaseAddr]
        add     DX,6
        in      AX,DX
        mov     [StartFrame],AX
; Ожидать завершения операции
@@Wait_OpComplete:
        ; Проверить номер кадра
        in      AX,DX
        sub     AX,[StartFrame]
        and     AX,7FFh   ;выделить младшие 11 разрядов
        cmp     AX,500    ;ожидать не более 500 кадров
        ja      @@TimeOut ;превышен интервал ожидания
        ; Проверить слово состояния
        cmp     [dword ptr GS:ESI],1b
        jne     @@Wait_OpComplete
        popad
        ret
; Превышен допустимый интервал ожидания
@@TimeOut:
        ; Остановить контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
        MFatalError TmOut
; Переполнен массив дескрипторов
@@TD_Array_Error:
        ; Остановить контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
        MFatalError DsErr
ENDP StatusIN_Transaction

;*****************************************************
;*            ПЕРЕДАТЬ КОМАНДУ УСТРОЙСТВУ USB        *
;* Передаваемые параметры:                           *
;* SI - смещение дескриптора команды в сегменте данных.*
;*****************************************************
PROC Setup_Transaction near
        pushad
; Вычислить линейный адрес данных дескриптора команды
        xor     EAX,EAX
        mov     AX,DS
        shl     EAX,4
        and     ESI,0FFFFh
        add     EAX,ESI
        mov     [Addr_CommandDescr],EAX
```

продолжение

Листинг 8.1 (*продолжение*)

```
; Загрузить в ESI указатель на массив дескрипторов
        mov     ESI,[Addr_TD_Array]
; Сформировать дескриптор команды
        ; Указатель на следующий TD
        mov     EAX,ESI
        add     EAX,32
        mov     [GS:ESI],EAX
        ; Слово управления
        mov     EAX,[ShDevType] ;тип устройства
        or      EAX,00800000h   ;признак активности
        mov     [GS:ESI+4],EAX
        ; Маркер
        mov     EAX,[ShFuncNum] ;номер функции
        or      EAX,00E0002Dh   ;передать 8 байт
        mov     [GS:ESI+8],EAX
        ; Указатель на дескриптор команды
        mov     EAX,[Addr_CommandDescr]
        mov     [GS:ESI+12],EAX
        xor     EAX,EAX
        mov     [GS:ESI+16],EAX
        mov     [GS:ESI+20],EAX
        mov     [GS:ESI+24],EAX
        mov     [GS:ESI+28],EAX
        add     ESI,32

; Сформировать дескриптор для приема пустого пакета
        mov     [dword ptr GS:ESI],1b ;последний TD
        ; Слово управления
        mov     EAX,[ShDevType] ;тип устройства
        or      EAX,00800000h   ;признак активности
        mov     [GS:ESI+4],EAX
        ; Маркер
        mov     EAX,[ShFuncNum] ;номер функции
        or      EAX,0FFE80069h  ;принять пустой блок
        mov     [GS:ESI+8],EAX
        xor     EAX,EAX
        mov     [GS:ESI+12],EAX
        mov     [GS:ESI+16],EAX
        mov     [GS:ESI+20],EAX
        mov     [GS:ESI+24],EAX
        mov     [GS:ESI+28],EAX
; Установить указатель на список дескрипторов
; (контроллер начинает передачу данных)
        mov     EAX,[Addr_TD_Array]
        mov     ESI,[Addr_QH]
        add     ESI,4
        mov     [GS:ESI],EAX
; Запомнить номер текущего кадра
        mov     DX,[USB_BaseAddr]
```

```
        add     DX,6
        in      AX,DX
        mov     [StartFrame],AX
; Ожидать завершения операции
@@Wait_OpComplete:
        ; Проверить номер кадра
        in      AX,DX
        sub     AX,[StartFrame]
        and     AX,7FFh   ;выделить младшие 11 разрядов
        cmp     AX,500    ;ожидать не более 500 кадров
        ja      @@TimeOut ;превышен интервал ожидания
        ; Проверить слово состояния
        cmp     [dword ptr GS:ESI],1b
        jne     @@Wait_OpComplete
        popad
        ret
; Превышен допустимый интервал ожидания
@@TimeOut:
        ; Остановить контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
        MFatalError TmOut
ENDP Setup_Transaction

;************************************************
;* ПРИСВОИТЬ ПОРЯДКОВЫЙ НОМЕР НАЙДЕННОМУ УСТРОЙСТВУ *
;* Передаваемые параметры:                        *
;* USB_PortReg - адрес регистра порта;            *
;* USB_Device_Number - порядковый номер устройства. *
;************************************************
PROC Enumeration near
        pushad
; Разблокировать порт
        mov     DX,[USB_PortReg]
        mov     AX,1110b
        out     DX,AX
; Определить и запомнить тип устройства
        mov     DX,[USB_PortReg]
        in      AX,DX
        test    AX,100000000b
        jnz     @@Low
        mov     [dword ptr ShDevType],0
        jmp short @@DeviceTypeSaved
@@Low:  mov     [dword ptr ShDevType],4000000h
@@DeviceTypeSaved:

; После сброса устройство имеет нулевой номер
        mov     [dword ptr ShFuncNum],0
; Вычислить порядковый номер
```

Листинг 8.1 (*продолжение*)

```
        inc     [USB_Device_Number]
; Подать команду "Set Address"
        xor     AX,AX
        mov     AL,[USB_Device_Number]
        mov     [word ptr SetAddrDesc+2],AX
        mov     SI,offset SetAddrDesc
        call    Setup_Transaction
; Присвоить устройству порядковый номер
        xor     EAX,EAX
        mov     AL,[USB_Device_Number]
        shl     EAX,8
        mov     [ShFuncNum],EAX
        popad
        ret
ENDP Enumeration

;************************************
;* ПОЛУЧИТЬ ДЕСКРИПТОР КОНФИГУРАЦИИ *
;************************************
PROC GetConfigurationDescriptor near
        pusha
; Подать команду "Get Configuration Descriptor"
        mov     SI,offset GetConfDesc
        call    StatusIN_Transaction
; Запомнить полный размер группы дескрипторов
; конфигурации
        mov     AX,[word ptr DataBuffer+2]
        mov     [word ptr GetConfDesc+6],AX
; Подать команду "Get Configuration Descriptor"
        mov     SI,offset GetConfDesc
        call    StatusIN_Transaction
        popa
        ret
ENDP GetConfigurationDescriptor
ENDS
```

Листинг 8.2 содержит программу USB_Device_Search, предназначенную для тестирования интерфейса USB. Программа выполняет поиск устройства USB по всем портам всех имеющихся в системе хост-контроллеров. Как только программа обнаруживает какое-либо устройство, она прекращает процесс поиска и выводи на экран содержимое регистров хост-контроллера, к которому подсоединено найденное устройство. Далее программа запрашивает у устройства и отображает на экран содержимое дескриптора устройства и дескриптора конфигурации.

Программа USB_Device_Search использует универсальные процедуры ввода-вывода из листинга 1.2, процедуру переключения в линейный

режим адресации из листинга 2.1, процедуры для вывода десятичных чисел из листинга 2.5 и набор процедур для работы с хост-контроллером и устройствами USB из листинга 8.1.

ПРИМЕЧАНИЕ

Программа может обнаружить только устройство с включенным электропитанием.

Листинг 8.2. Поиск устройств на шине USB и считывание параметров первого обнаруженного устройства

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Физический адрес области памяти для списка кадров USB
FrameListBaseAddr equ 200000h

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; ТЕКСТОВЫЕ СООБЩЕНИЯ
Txt0 DB LIGHTCYAN,0,26
     DB "ПОИСК И ОПРОС УСТРОЙСТВА USB",0
Txt1 DB 2,27,"ПАРАМЕТРЫ ХОСТ-КОНТРОЛЛЕРА",0
     DB 4,10,"Порядковый номер контроллера:",0
     DB 5,8,"Базовый адрес набора регистров:",0
     DB 6,8,"Номер используемого прерывания:",0
     DB 9,24,"СОДЕРЖИМОЕ РЕГИСТРОВ КОНТРОЛЛЕРА",0
     DB 11,23,"Регистр команды:",0
     DB 12,21,"Регистр состояния:",0
     DB 13,7,"Регистр управления прерываниями:",0
     DB 14,27,"Номер кадра:",0
     DB 15,11,"Базовый адрес списка кадров:",0
     DB 16,14,"Модификация начала кадра:",0
     DB 17,13,"Регистр состояния порта 1:",0
     DB 18,13,"Регистр состояния порта 2:",0
     DB 20,0,"Устройство подключено к порту N",0
     DB 22,0,"Тип устройства:",0
Full DB LIGHTCYAN,22,16,"полноскоростное",0
LowS DB LIGHTMAGENTA,22,16,"низкоскоростное",0
Txt2 DB 2,23,"СТАНДАРТНЫЙ ДЕСКРИПТОР УСТРОЙСТВА",0
     DB 4,14,"Размер дескриптора, байт:",0
     DB 5,18,"Код типа дескриптора:",0
```

продолжение

Листинг 8.2 (*продолжение*)

```
     DB 6,9,"Номер версии спецификации USB:",0
     DB 7,17,"Код класса устройства:",0
     DB 8,14,"Код подкласса устройства:",0
     DB 9,14,"Код протокола устройства:",0
     DB 10,1,"Макс. размер пакета для нулевой точки:",0
     DB 11,22,"Код изготовителя:",0
     DB 12,27,"Код изделия:",0
     DB 13,10,"Число доступных конфигураций:",0
Txt3 DB 2,23,"СТАНДАРТНЫЙ ДЕСКРИПТОР КОНФИГУРАЦИИ",0
     DB 4,14,"Размер дескриптора, байт:",0
     DB 5,18,"Код типа дескриптора:",0
     DB 6,14,"Общий объем данных, байт:",0
     DB 7,16,"Количество интерфейсов:",0
     DB 8,15,"Код данной конфигурации:",0
     DB 9,11,"Характеристики конфигурации:",0
     DB 10,13,"Потребляемая мощность, мА:",0
AnyK DB YELLOW,24,29,"Нажмите любую клавишу",0
; ДЕСКРИПТОРЫ КОМАНД
; Дескриптор команды "Get Device Descriptor"
GetDevDesc   DB 80h,6
             DW 100h,0,8
; Создать дескриптор команды "Set Address"
SetAddrDesc  DB 0,5
             DW 1,0,0
; Дескриптор команды "Get Configuration Descriptor"
GetConfDesc  DB 80h,6
             DW 200h,0,8
ENDS

; Область памяти для хранения дескрипторов передачи
SEGMENT USB_DESCR para public 'DATA'
; Заголовок очереди дескрипторов
QH_Descriptor DD  00000003h   ;единственный заголовок
              DD  00000000h   ;указатель на первый TD
              DD  0,0,0,0,0,0 ;область данных ПО
; Список дескрипторов для одной транзакции
TD_Array      DD  8*64 DUP(?)
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC USB_Device_Search
        mov     AX,DGROUP
```

```
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Проверить наличие PCI BIOS
        mov     AX,0B101h
        int     1Ah
        jc      @@PCIBIOSNotFound
        cmp     EDX,20494350h
        jne     @@PCIBIOSNotFound
; Установить режим прямой адресации памяти
        call    Initialization
; Инициализировать дескрипторы USB
        call    InitializeDeascriptors

; ЦИКЛ ПОИСКА ХОСТ-КОНТРОЛЛЕРОВ
        mov     [USB_HostIndex],0
@@NextHost:
; Найти контроллер USB
        call    FindUSBController
        cmp     [SearchResult],0
        jne     @@NoHost
; Произвести глобальный сброс контроллера
        mov     DX,[USB_BaseAddr]
        mov     AX,100b         ;установить сигнал сброса
        out     DX,AX
; Ожидать не менее 10 мс
        call    Wait05s
; Снять сигнал сброса
        mov     AX,0
        out     DX,AX
; Ожидать не менее 10 мс
        call    Wait05s
; Проверить регистр состояния порта 1
        mov     [USB_PortNum],1
        mov     DX,[USB_BaseAddr]
        add     DX,10h
        in      AX,DX
        test    AX,000Fh
        jnz     @@SavePortReg
; Проверить регистр состояния порта 2
        mov     [USB_PortNum],2
        add     DX,2
        in      AX,DX
        test    AX,000Fh
```

продолжение

Листинг 8.2 (*продолжение*)

```
        jz      @@NextHost
; Запомнить адрес регистра состояния порта
@@SavePortReg:
        mov     [USB_PortReg],DX
; Загрузить указатель на список кадров в регистр
; адреса списка кадров
        mov     DX,[USB_BaseAddr]
        add     DX,6
        mov     AX,0
        out     DX,AX
        add     DX,2
        mov     EAX,FrameListBaseAddr
        out     DX,EAX

; ОТОБРАЗИТЬ СОДЕРЖИМОЕ РЕГИСТРОВ ВВОДА-ВЫВОДА
; Очистить экран
        call    ClearScreen
; Вывести заголовок экрана
        MShowColorString Txt0
; Вывести заголовки полей
        mov     [TextColorAndBackground],LIGHTGREEN
        MShowText 15,Txt1
; Вывести базовый адрес и номер прерывания
        mov     [TextColorAndBackground],LIGHTGREY
        MShowDecByte 4,40,<[byte ptr USB_HostIndex]>
        MShowHexWord 5,40,[USB_BaseAddr]
        MShowHexByte 6,40,[USB_IntLine]
; Вывести содержимое регистров
        mov     DX,[USB_BaseAddr]
        in      AX,DX
        MShowHexWord 11,40,AX
        add     DX,2
        in      AX,DX
        MShowHexWord 12,40,AX
        add     DX,2
        in      AX,DX
        MShowHexWord 13,40,AX
        add     DX,2
        in      AX,DX
        MShowHexWord 14,40,AX
        add     DX,2
        in      EAX,DX
        MShowHexDWord 15,40,EAX
        add     DX,4
        in      AL,DX
        MShowHexByte 16,40,AL
        mov     DX,[USB_BaseAddr]
        add     DX,10h
        in      AX,DX
```

```
        MShowHexWord 17,40,AX
        add     DX,2
        in      AX,DX
        MShowHexWord 18,40,AX

; Показать номер используемого порта
        MShowDecByte 20,32,[USB_PortNum]
; Указать тип устройства
        mov     DX,[USB_PortReg]
        in      AX,DX
        test    AX,100000000b
        jnz     @@Low
        mov     [dword ptr ShDevType],0
        MShowColorString Full
        jmp short @@WaitAnyKey
@@Low:  mov     [dword ptr ShDevType],4000000h
        MShowColorString LowS

; Ожидать нажатия любой клавиши
@@WaitAnyKey:
        MShowColorString AnyK
        call    GetChar

; Включить хост-контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,1
        out     DX,AX
; Присвоить порядковый номер найденному устройству
        mov     [USB_Device_Number],0
        call    Enumeration

; ПОЛУЧИТЬ И ОТОБРАЗИТЬ НА ЭКРАНЕ ДЕСКРИПТОР УСТРОЙСТВА
; Подать команду "Get Device Descriptor"
        mov     SI,offset GetDevDesc
        call    StatusIN_Transaction
; Очистить экран
        call    ClearScreen
; Вывести заголовок экрана
        MShowColorString Txt0
; Вывести заголовки полей
        mov     [TextColorAndBackground],LIGHTGREEN
        MShowText 11,Txt2
; Вывести базовый адрес и номер прерывания
        mov     [TextColorAndBackground],LIGHTGREY
        MShowDecByte  4,40,[DataBuffer]
        MShowHexByte  5,40,<[DataBuffer+1]>
        MShowHexWord  6,40,<[word ptr DataBuffer+2]>
        MShowHexByte  7,40,<[DataBuffer+4]>
        MShowHexByte  8,40,<[DataBuffer+5]>
        MShowHexByte  9,40,<[DataBuffer+6]>
```

продолжение

Листинг 8.2 (*продолжение*)

```
        MShowDecByte 10,40,<[DataBuffer+7]>
        MShowHexWord 11,40,<[word ptr DataBuffer+8]>
        MShowHexWord 12,40,<[word ptr DataBuffer+10]>
        MShowDecByte 13,40,<[DataBuffer+17]>
; Ожидать нажатия любой клавиши
        MShowColorString AnyK
        call    GetChar

; ПОЛУЧИТЬ И ОТОБРАЗИТЬ НА ЭКРАНЕ ДЕСКРИПТОР КОНФИГУРАЦИИ
        call    GetConfigurationDescriptor
; Очистить экран
        call    ClearScreen
; Вывести заголовок экрана
        MShowColorString Txt0
; Вывести заголовки полей
        mov     [TextColorAndBackground],LIGHTGREEN
        MShowText 8,Txt3
; Вывести базовый адрес и номер прерывания
        mov     [TextColorAndBackground],LIGHTGREY
        MShowDecByte  4,40,[DataBuffer]
        MShowHexByte  5,40,<[DataBuffer+1]>
        MShowDecWord  6,40,<[word ptr DataBuffer+2]>
        MShowDecByte  7,40,<[DataBuffer+4]>
        MShowHexByte  8,40,<[DataBuffer+5]>
        MShowBinByte  9,40,<[DataBuffer+7]>
        xor     AX,AX
        mov     AL,[DataBuffer+8]
        shl     AX,2
        MShowDecWord  10,40,AX
; Ожидать нажатия любой клавиши
        MShowColorString AnyK
        call    GetChar

; ВЫКЛЮЧИТЬ ХОСТ-КОНТРОЛЛЕР
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h

; ОБРАБОТКА ОШИБОК
; Не поддерживается PCI BIOS
@@PCIBIOSNotFound:
        MFatalError NoPCI
; Не найден хост или устройство
```

```
@@NoHost:
        cmp     [USB_HostIndex],0
        je      @@HostNotFound
; Устройство USB не найдено
@@DeviceNotFound:
        MFatalError NoDev
; Нет ни одного контроллера USB
@@HostNotFound:
        MFatalError NoUSB
ENDP USB_Device_Search
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить процедуры перевода десятичных чисел
include "list2_05.inc"
; Подключить процедуры для работы с контроллером USB
include "listB_01.inc"
END
```

Взаимодействие хост-контроллера с хабом

Чтобы получить доступ к устройствам, подключенным к шине USB через хаб, хост должен сконфигурировать хаб и произвести настройку его портов. Процесс настройки хаба хост осуществляет путем подачи определенной последовательности запросов.

Прежде всего хост должен произвести идентификацию устройства по стандартным дескрипторам. Хаб можно опознать по дескрипторам устройства и интерфейса.

- В Стандартном дескрипторе устройства поле кода класса устройства содержит значение 09h, поле кода подкласса устройства — значение 00h.
- В Стандартном дескрипторе интерфейса поле количества конечных точек имеет значение 01h, поле кода класса интерфейса содержит значение 09h, поле кода подкласса устройства — значение 00h, поле протокола — значение 00h.

Хаб имеет только одну входную конечную точку, которая работает в режиме передачи по прерываниям. Поле значения интервала обслуживания в дескрипторе конечной точки содержит значение FFh.

Время реакции хаба на стандартные запросы не должно превышать 50 мс. Хаб поддерживает следующие стандартные запросы: Get Status, Clear Feature, Set Feature, Set Address, Get Descriptor, Set Descriptor, Get Configuration, Set Configuration. Реакция на запросы Get Interface и Set Interface не определена, так как хаб может иметь только один интерфейс; реакция на запрос Synch Frame не определена, так как хаб не имеет изохронных конечных точек.

Дескриптор хаба

Кроме стандартных дескрипторов, по запросу может быть выдан специфический Дескриптор хаба (Hub Descriptor), структура которого приведена в табл. 8.9.

Таблица 8.9. Структура Дескриптора хаба

Смещение	Мнемоника	Размер	Описание
0	bDescLength	BYTE	Размер данного дескриптора в байтах
1	bDescriptorType	BYTE	Тип дескриптора (29h)
2	bNbrPorts	BYTE	Количество нисходящих портов
3	wHubCharacteritics	WORD	Характеристики хаба: биты 0–1 — логический режим управления энергией (00b — подача энергии включаются одновременно для всех портов, 01b — возможно индивидуальное управление подачей энергии для каждого порта, 10b или 11b — у хаба отсутствует схема управления энергией); бит 2 — признак составного устройства (0 — хаб является самостоятельным устройством, 1 — хаб входит в состав периферийного устройства); биты 3–4 — режим защиты от перегрузки (00b — глобальная защита от перегрузки по сумме токов всех портоа, 01b — индивидуальнвя защита для каждого порта, 10b или 11b — защита отсутствует); биты 5–15 зарезервированы
5	bPwrOn2PwrGood	BYTE	Величина временного интервала от подачи команды включения энергии до стабилизации напряжения питания на выходе порта, заданная с шагом 2 мс

Смещение	Мнемоника	Размер	Описание
6	bHubContr-Current	BYTE	Величина тока (мА), потребляемого контроллером хаба
7	Device-Removable	K байт	Битовая карта подключения съемных устройств к портам: бит 0 зарезервирован, бит 1 соответствует порту 1, бит 2 — порту 2, j, бит n — порту n. Если бит имеет значение 0, к порту подключено съемное устройство, если 1 —несъемное
7 + K	PortPwr-CtrlMask	K байт	Маска контроля питания портов (каждому порту соответствует один бит маски). Маска входит в дескриптор для сохранения совместимости со стандартом USB 1.0 и в настоящее время не используется (все разряды установлены в 1)

Запросы, специфические для хабов

В спецификации USB для хабов определены следующие коды специфических запросов:

- 0 — GET_STATUS (определить состояние устройства);
- 1 — CLEAR_FEATURE (сбросить свойство);
- 2 — GET_STATE (получить состояние устройства);
- 3 — SET_FEATURE (установить свойство);
- 6 — GET_DESCRIPTOR (получить дескриптор);
- 7 — SET_DESCRIPTOR (загрузить дескриптор).

Кроме того, для хабов в спецификации USB определены следующие значения селектора свойств:

- 0 — C_HUB_LOCAL_POWER (признак изменения состояния встроенного источника питания),
- 1 — C_HUB_OVER_CURRENT (признак изменения состояния индикатора перегрузки по току).

Для портов хабов в спецификации USB определены следующие значения селектора свойств:

- 0 — PORT_CONNECTION (к порту подключено устройство),
- 1 — PORT_ENABLE (работа порта разрешена),

- 2 — PORT_SUSPEND (порт находится в состоянии ожидания),
- 3 — PORT_OVER_CURRENT (перегрузка по току),
- 4 — PORT_RESET (установлен сигнал сброса),
- 8 — PORT_POWER (питание включено),
- 9 — PORT_LOW_SPEED (порт работает в низкоскоростном режиме),
- 16 — C_PORT_CONNECTION (признак изменения состояния подключения),
- 17 — C_PORT_ENABLE (признак выполнения операции разрешения или запрета работы порта),
- 18 — C_PORT_ SUSPEND (признак переключения из состояния ожидания в активный режим или наоборот),
- 19 — C_PORT_OVER_CURRENT (признак изменения состояния индикатора перегрузки по току),
- 20 — C_PORT_RESET (признак того, что сигнал сброса был установлен или снят).

Запрос Clear Hub Feature используется для того, чтобы сбросить признак состояния хаба, указанный значением селектора свойств. Запрос имеет следующие параметры:

- bmRequestType = 00100000b;
- bRequest = CLEAR_FEATURE (01h);
- wValue — селектор свойств хаба;
- wIndex = 0;
- wLength = 0.

Передача данных по запросу Clear Hub Feature не производится.

Запрос Clear Port Feature используется для того, чтобы сбросить состояние порта хаба, указанное значением селектора свойств. Запрос имеет следующие параметры:

- bmRequestType = 00100011b;
- bRequest = CLEAR_FEATURE (01h);
- wValue — селектор свойств порта;
- wIndex — номер порта;
- wLength = 0.

Передача данных по запросу Clear Port Feature не производится.

Запрос Clear Port Feature допускает использование следующих селекторов: PORT_ENABLE, PORT_SUSPEND, PORT_POWER, C_PORT_CONNECTION, C_PORT_ENABLE, C_PORT_SUSPEND, C_PORT_OVER_CURRENT, C_PORT_RESET.

Сброс свойства PORT_SUSPEND вызывает формирование сигнала пробуждения для порта, который находится в состоянии ожидания.

После сброса свойства PORT_ENABLE работа порта будет запрещена.

Сброс свойства PORT_POWER вызывает отключение питания порта.

Запрос Get Bus State используется для диагностики порта с заданным номером. Запрос имеет следующие параметры:

- bmRequestType = 10100011b;
- bRequest = GET_STATE (02h);
- wValue = 0;
- wIndex — номер порта;
- wLength = 1.

По запросу Get Bus State хаб возвращает один байт данных со следующей структурой:

- бит 0 — значение сигнала на линии D–;
- бит 1 — значение сигнала на линии D+;
- биты 2–7 зарезервированы и должны быть сброшены в 0.

Запрос Get Hub Descriptor позволяет хосту получить дескриптор хаба. Запрос имеет следующие параметры:

- bmRequestType = 10100000b;
- bRequest = GET_DESCRIPTOR (06h);
- wValue содержит тип дескриптора (29h) в старшем байте и индекс дескриптора (00h в младшем байте);
- wIndex = 0;
- wLength — размер дескриптора в байтах.

По запросу Get Hub Descriptor хаб возвращает дескриптор, структура которого описана в табл. 8.9.

Запрос Get Hub Status позволяет определить текущее состояние хаба. Запрос имеет следующие параметры:

- bmRequestType = 10100000b;
- bRequest = GET_STATUS (00h);
- wValue = 0;
- wIndex = 0;
- wLength = 4.

По запросу Get Hub Status хаб возвращает 16-разрядное слово состояния wHubStatus и 16-разрядное слово индикаторов изменения состояния wHubChange.

Слово состояния хаба имеет следующую структуру:

- бит 0 содержит признак неисправности встроенного источника питания хаба (0 — встроенный источник питания находится в рабочем состоянии, 1 — источник выключен, и хаб получает питание от шины USB);
- бит 1 содержит признак перегрузки порта по выходному току (0 — порт работает нормально, 1 — подключенное к порту устройство потребляет слишком много энергии);
- биты 2–15 зарезервированы (содержат нули).

Слово индикаторов изменения состояния хаба имеет следующую структуру:

- бит 0 содержит признак изменения состояния встроенного источника питания C_HUB_LOCAL_POWER (устанавливается в 1, если состояние встроенного источника питания изменилось);
- бит 1 содержит признак изменения состояния индикатора перегрузки по току C_HUB_OVER_CURRENT (устанавливается в 1, если состояние индикатора перегрузки изменилось);
- биты 2–15 зарезервированы (содержат нули).

Запрос Get Port Status позволяет определить текущее состояние заданного порта хаба. Запрос имеет следующие параметры:

- bmRequestType = 10100011b;
- bRequest = GET_STATUS (00h);
- wValue = 0;
- wIndex — номер порта;
- wLength = 4.

По запросу Get Port Status хаб возвращает 16-разрядное слово состояния порта wPortStatus и 16-разрядное слово индикаторов изменения состояния порта wPortChange.

Слово состояния порта имеет следующую структуру:

- бит 0 содержит признак подключения PORT_CONNECTION (0 — порт свободен, 1 — к порту подключено устройство);
- бит 1 содержит признак разрешения работы PORT_ENABLE (0 — работа порта запрещена, 1 — работа разрешена);
- бит 2 содержит признак состояния ожидания PORT_SUSPEND (устанавливается в 1, если порт находится в состоянии ожидания);
- бит 3 содержит индикатор перегрузки по току PORT_OVER_CURRENT (устанавливается в 1, если устройство потребляет от порта слишком большой ток);

- бит 4 содержит признак активности сигнала сброса PORT_RESET (устанавливается в 1, если на шине установлен сигнал сброса);
- биты 5–7 зарезервированы (содержат нули);
- бит 8 содержит индикатор состояния схемы управления энергией PORT_POWER (0 — питание на устройство не подается, 1 — подача питания разрешена);
- бит 9 содержит признак подключения низкоскоростного устройства PORT_LOW_SPEED (устанавливается в 1, если к порту подсоединено низкоскоростное устройство);
- биты 10–15 зарезервированы (содержат нули).

Слово индикаторов изменения состояния имеет следующую структуру:

- бит 0 содержит индикатор изменения состояния подключения C_PORT_CONNECTION (устанавливается в 1, если произошло подключение или отключение устройства);
- бит 1 содержит индикатор изменения значения признака разрешения работы порта C_PORT_ENABLE (устанавливается в 1, если работа порта была разрешена или запрещена);
- бит 2 содержит индикатор изменения значения признака состояния ожидания C_PORT_SUSPEND (устанавливается в 1, если порт был переключен из состояния ожидания в активный режим работы или наоборот);
- бит 3 содержит индикатор изменения состояния сигнала перегрузки по току C_PORT_OVER_CURRENT (устанавливается в 1, если состояние сигнала перегрузки изменилось);
- бит 4 содержит индикатор изменения значения признака активности сигнала сброса C_PORT_RESET (устанавливается в 1 после установки или снятия сигнала сброса);
- биты 5–15 зарезервированы (содержат нули).

Запрос Set Hub Descriptor позволяет хосту перезаписать (заменить) дескриптор хаба. Запрос имеет следующие параметры:

- bmRequestType = 00100000b;
- bRequest = SET_DESCRIPTOR (07h);
- wValue содержит тип дескриптора (29h) в старшем байте и индекс дескриптора (00h в младшем байте);
- wIndex = 0;
- wLength — размер дескриптора в байтах.

По запросу Set Hub Descriptor хост передает хабу дескриптор, структура которого описана в табл. 8.9.

Запрос Set Hub Feature используется для того, чтобы установить признак состояния хаба, указанный значением селектора свойств. Запрос имеет следующие параметры:

- bmRequestType = 00100000b;
- bRequest = SET_FEATURE (03h);
- wValue — селектор свойств хаба;
- wIndex = 0;
- wLength = 0.

Передача данных по запросу Set Hub Feature не производится.

Запрос Set Port Feature используется для того, чтобы перевести порт хаба в состояние, указанное значением селектора свойств. Запрос имеет следующие параметры:

- bmRequestType = 00100011b;
- bRequest = SET_FEATURE (03h);
- wValue — селектор свойств порта;
- wIndex — номер порта;
- wLength = 0.

Передача данных по запросу Set Port Feature не производится.

Запрос Set Port Feature допускает использование следующих селекторов: PORT_ENABLE, PORT_SUSPEND, PORT_POWER, C_PORT_CONNECTION, C_PORT_ENABLE, C_PORT_SUSPEND, C_PORT_OVER_CURRENT, C_PORT_RESET.

После установки свойства PORT_SUSPEND порт и подсоединенное к нему устройство переводятся в состоянии ожидания.

После установки свойства PORT_ENABLE работа порта будет разрешена.

После установки свойства PORT_POWER будет включено питание порта.

Процедура нумерации и конфигурирования устройств на шине USB

После включения питания хоста и подачи на шину USB сигнала сброса все устройства, которые подсоединены к шине и включены, находятся в несконфигурированном состоянии, имеют на шине адрес 0 и реагируют только на запросы по Основному каналу сооб-

щений. Хост должен поочередно идентифицировать каждое устройство, присвоить ему индивидуальный адрес и сконфигурировать его.

Конфигурирование системы начинается с подачи сигнала глобального сброса. Далее хост идентифицирует и конфигурирует устройства, подключенные непосредственно к портам хост-контроллера.

Для того чтобы определить, подключено ли к порту хост-контроллера какое-либо устройство, нужно разрешить работу порта, записав 1 во второй разряд регистра состояния и управления порта. После этого нужно прочитать регистр состояния порта и проверить значение нулевого разряда слова состояния: если к порту подключено устройство, этот разряд будет установлен в 1. Кроме того, по значению восьмого разряда слова состояния нужно определить тип устройства (0 — полноскоростное устройство, 1 — низкоскоростное).

Далее хост выполнит процедуру конфигурирования устройства. Порядок конфигурирования приведен ниже.

1. Хост подает запрос Set Address и присваивает обнаруженному устройству адрес на шине USB в соответствии с его порядковым номером.
2. Хост считывает дескриптор устройства и дескрипторы конфигурации.
3. Хост выбирает нужную конфигурацию устройства и устанавливает ее при помощи запроса Set Configuration.

ВНИМАНИЕ

Поскольку все устройства после сброса реагируют на нулевой адрес, разблокировать следующий порт можно только после завершения конфигурирования устройства, подключенного к текущему порту.

Операционные системы Windows 98 SE, Windows 2000 и Windows XP используют несколько более сложный порядок конфигурирования [39].

1. Хост подает команду сброса на порт, к которому подключено устройство.
2. Хост запрашивает дескриптор устройства, причем в запросе задает размер дескриптора 64 байта.
3. Хост принимает первые 8 байт дескриптора устройства, после чего подает команду сброса порта еще раз.
4. Хост присваивает устройству адрес с помощью запроса Set Address.

5. Хост снова считывает дескриптор устройства, причем в запросе указывается «правильный» размер дескриптора — 18 байт.
6. Хост запрашивает дескриптор конфигурации устройства, задавая в запросе размер дескриптора 9 байт, и извлекает из него значение полного размера списка дескрипторов.
7. Хост повторно запрашивает дескриптор конфигурации устройства и все сопутствующие ему дескрипторы, используя полученный размер списка.
8. Хост запрашивает дескрипторы строк, если они определены.

Если какое-либо из устройств, подключенных к порту, является хабом, хост продолжает процесс настройки системы, поочередно опрашивая каждый порт хаба. Каждому найденному устройству хост присваивает адрес и задает конфигурацию по описанной выше схеме. Процесс конфигурирования продолжается, пока не будут настроены все хабы и подсоединенные к ним периферийные устройства с включенным питанием (выключенные устройства на запросы не отвечают, поэтому хост их «не видит»).

В процессе работы некоторые устройства могут быть отсоединены от шины или подсоединены к ней. Хост должен периодически контролировать состояние собственных портов и портов хабов (путем опроса по прерываниям), чтобы контролировать изменения на шине USB. Каждому вновь подключенному устройству должен быть присвоен номер и задана конфигурация.

Работа с принтером через интерфейс USB

В процессе стандартизации периферийного оборудования с интерфейсом USB постоянно возникают проблемы, связанные с несогласованностью действий разработчиков, а часто — и с явным нежеланием следовать каким-либо стандартам. Некоторые классы устройств описаны настолько расплывчато, что создание для них универсальных драйверов в принципе невозможно: для разработки программного обеспечения требуется запрашивать у изготовителя документацию на конкретное устройство.

Принтеры являются счастливым исключением из общего правила: интерфейс USB для принтеров разработан таким образом, чтобы имитировать работу с принтером через параллельный порт, и полностью стандартизирован. Описание интерфейса принтеров приво-

дится в спецификации Universal Serial Bus Device Class Definition for Printing Devices [93].

Для управления процессом печати используются командные языки фирм-изготовителей принтеров, поэтому создать единый универсальный драйвер практически невозможно. Существуют, однако, достаточно крупные группы изделий, для которых можно создать специфические драйверы, используя открытые (опубликованные) спецификации на командные языки. Например, как уже было указано в главе «Принтеры: печать в растровом режиме», для лазерных принтеров фактическим стандартом является язык HP PCL, а все струйные принтеры EPSON поддерживают язык Epson raster. Многие модели лазерных принтеров поддерживают также язык PostScript.

В обязательном порядке принтер имеет по крайней мере одну выходную точку, работающую в режиме передачи массивов данных (Bulk OUT). Эта точка служит для передачи данных (печатаемого текста или изображения) с хоста на принтер.

Принтер может иметь также входную точку, работающую в режиме передачи массивов данных (Bulk IN) и предназначенную для передачи хосту информации об устройстве (например, о наличии бумаги в лотке подачи и тонера в картридже) и текущем состоянии процесса печати документа (например, об используемом разрешении, режиме печати, применяемых шрифтах).

Любой принтер с интерфейсом USB должен поддерживать по крайней мере один из двух возможных интерфейсов:

- однонаправленный интерфейс (Unidirectional Interface) поддерживает только передачу данных с хоста на принтер через выходную точку. Данные о состоянии принтера передаются через Основной канал сообщений по запросу GET_PORT_STATUS;
- двунаправленный интерфейс (Bi-directional Interface) поддерживает передачу данных с хоста на принтер через выходную точку и позволяет хосту получать информацию о принтере и состоянии процесса печати через входную точку. Данные о текущем состоянии принтера можно также получить через Основной канал сообщений по запросу GET_PORT_STATUS.

Обычно используется только двунаправленный интерфейс. Если принтер поддерживает одновременно оба интерфейса, они должны быть реализованы как альтернативные. Тип интерфейса указывается в поле протокола дескриптора интерфейса: однонаправленному интерфейсу соответствует код 01h, двунаправленному — код 02h.

Принтеры поддерживают все стандартные запросы к устройству USB, а также несколько специфических (для своего класса) запросов:

- Get Device ID – получить идентификатор устройства;
- Get Port Status – получить информацию о текущем состоянии принтера;
- Soft Reset – произвести программный сброс принтера.

Запрос Get Device ID позволяет получить строку-идентификатор принтера. Запрос имеет следующие параметры:

- bmRequestType = 10100001b;
- bRequest = 0;
- wValue — индекс конфигурации (нумерация конфигураций начинается с нуля);
- wIndex — код интерфейса (в старшем байте — индекс интерфейса, в младшем байте — индекс варианта настройки; индексация начинается с нуля);
- wLength — предельная длина возвращаемой строки в байтах (ограничитель на длину строки).

По запросу Get Device ID принтер передает хосту строку-идентификатор (Device ID) в формате, соответствующем стандарту IEEE-1284: первые два байта содержат 16-разрядное слово, задающее общую длину идентификатора в байтах, а вслед за ними размещается собственно строка-идентификатор в коде ASCII.

ПРИМЕЧАНИЕ

При расчете размера идентификатора в байтах учитываются длина поля размера (2 байта) и длина текстовой строки (строка не имеет оконечного ограничителя и состоит исключительно из символов ASCII).

Принтеры, имеющие несколько конфигураций, интерфейсов или несколько альтернативных установок для интерфейса, могут выдавать отдельную строку-идентификатор для каждого из возможных вариантов настройки.

Запрос Get Port Status позволяет получить содержимое регистра состояния принтера. Запрос имеет следующие параметры:

- bmRequestType = 10100001b;
- bRequest = 1;
- wValue = 0;
- wIndex — индекс интерфейса;
- wLength = 1.

По запросу Get Port Status принтер передает хосту байт состояния, который имеет формат, аналогичный изображенному на рис. 7.2 формату регистра состояния параллельного порта. Байт состояния имеет следующую структуру:

- биты 0–2 не используются, установлены в 0;
- бит 3 — признак ошибки ввода-вывода (0 — ошибка, 1 — нет ошибки);
- бит 4 — признак выбора принтера (0 — принтер в автономном режиме, 1 — принтер в режиме подключения);
- бит 5 — контроль наличия бумаги (0 — бумага вставлена, 1 — нет бумаги);
- биты 6 и 7 не используются, установлены в 0.

Таким образом, для передачи информации используются только три разряда байта состояния, а остальные разряды зарезервированы.

По запросу Soft Reset производится программный сброс принтера. Запрос имеет следующие параметры:

- bmRequestType = 00100011b;
- bRequest = 2;
- wValue = 0;
- wIndex — индекс интерфейса;
- wLength = 0.

Передача данных при выполнении запроса не производится.

Запрос Soft Reset вызывает очистку всех буферов данных принтера и сброс входного (Bulk IN) и выходного (Bulk OUT) каналов передачи массивов данных; все признаки ошибок и сбоев также сбрасываются. На USB-адрес и конфигурацию устройства программный сброс не влияет.

Устройство, относящееся к классу принтеров, должно поддерживать стандартные дескрипторы устройства, конфигурации, интерфейса и конечных точек.

Поля bDeviceClass, bDeviceSubClass, bDeviceProtocol в дескрипторе устройства содержат нули и не могут использоваться для проверки принадлежности устройства к классу принтеров. Поля idVendor и idProduct пригодны для идентификации устройства только в том случае, если программисту известно значение этих полей для принтеров данного типа: списки числовых идентификаторов для изделий известных фирм можно найти в Интернете; идентификатор устройства также иногда указывается в фирменной документации по программированию.

Разбор структуры данных, возвращаемой по запросу дескриптора конфигурации, начинается с самого дескриптора конфигурации. Из дескриптора конфигурации нужно извлечь и запомнить общую длину списка дескрипторов: в процессе просмотра списка это значение используется как признак конца списка.

В дескрипторе интерфейса требуется проверить значение полей класса и подкласса устройства: для принтеров код базового класса имеет значение 07h, код подкласса — 01h. Тип интерфейса (однонаправленный или двунаправленный) особой роли не играет, так как программист обычно может работать только с Основным каналом сообщений и выходным каналом: структура информации о принтере, которую хост получает от входной точки, известна только фирме-разработчику устройства (открыто не публикуется).

Из дескриптора выходной конечной точки (Bulk OUT) нужно извлечь значение поля bEndpointAddress, содержащего адрес этой конечной точки, и значение поля wMaxPacketSize, задающего максимальный размер пакета при передаче данных на принтер. Опознать дескриптор конечной точки можно по значению полей bLength (поле должно содержать значение 07h), bDescriptorType (поле должно содержать значение 05h), bEndpointAddress (поле должно содержать 0 в старшем разряде) и bmAttributes (поле должно содержать значение 02h).

В листинге 8.3 приведена программа USB_EpsonStylus_Test, осуществляющая печать заштрихованного квадрата на струйном принтере EPSON Stylus в растровом черно-белом режиме с разрешением 360×360 точек/дюйм. Программа использует следующие процедуры:

- процедура BulkOUT_Transaction передает на принтер команды и растровое изображение в режиме передачи массивов данных;
- процедура OutCharToPrn осуществляет вывод байта данных в промежуточный буфер (во избежание переполнения накопленную в буфере информацию нужно периодически сбрасывать на принтер при помощи процедуры BulkOUT_Transaction);
- процедура OutCommandToPrn использует подпрограмму OutCharToPrn для подачи на принтер командной последовательности символов;
- процедура Show_Ident выводит на экран строку-идентификатор принтера.

Кроме того, программа USB_EpsonStylus_Test использует универсальные процедуры ввода-вывода из листинга 1.2, процедуру переключения в линейный режим адресации из листинга 2.1 и набор процедур для работы с контроллером и устройствами USB из листинга 8.1.

ПРИМЕЧАНИЕ

Программа USB_EpsonStylus_Test осуществляет поиск принтера только непосредственно по портам хост-контроллера, поэтому перед запуском теста нужно подсоединить принтер к одному из USB-портов системного блока.

Листинг 8.3. Печать заштрихованного квадрата на струйном принтере EPSON Stylus с интерфейсом USB

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Физический адрес области памяти для списка кадров USB
FrameListBaseAddr equ 200000h

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

DATASEG
; Текстовые сообщения
Txt0 DB LIGHTMAGENTA,0,22
     DB "ПОИСК И ТЕСТИРОВАНИЕ ПРИНТЕРА EPSON",0
     DB YELLOW,11,19
     DB "Включите принтер, установите лист бумаги в",0
     DB YELLOW,12,13,"приемный лоток и нажмите "
     DB "любую клавишу на клавиатуре.",0
Txt1 DB LIGHTGREEN,11,0,"Идентификатор принтера:",0
     DB YELLOW,24,9,"Нажмите любую клавишу на "
     DB "клавиатуре и ждите завершения печати",0
; Сообщения об ошибках
NoPrn DB 12,22,"Струйный принтер EPSON не обнаружен",0
PrnEr DB 12,27,"Принтер не готов к работе",0
; Номер печатаемой строки изображения
PrintingString    DW ?
; Номер печатаемого байта
PrintingByte      DW ?

; КОМАНДЫ ДЛЯ ПРИНТЕРА
; Выйти из "Packet Mode"
ExitPacketMode DB 1Bh,0,0,0,1Bh,1
               DB "@EJL 1284.4",0Ah,"@EJL     ",0Ah
; Выйти из "Remote Mode"
TerminateRemoteMode DB 4, 1Bh,0,0,0
; Инициализировать принтер
PrnInitialization  DB 2, 1Bh,'@'
```

продолжение

Листинг 8.3 (*продолжение*)

```
; Установить графический режим
SelectGraphicsMode  DB 6, 1Bh,'(','G',1,0,1
; Выбор монохромного режима
MonochromeSelection DB 7, 1Bh,'(','K',1,0,0,1
; Печать растровой графики (320 точек в строке)
PrintRasterData     DB 8, 1Bh,'.',0,10,10,1,64,1
; Перевод строки
SetRelVertPosition  DB 7, 1Bh,'(','v',2,0,1,0

; ДЕСКРИПТОРЫ КОМАНД
; Дескриптор команды "Get Device Descriptor"
GetDevDesc    DB 80h,6
              DW 100h,0,8
; Дескриптор команды "Set Address"
SetAddrDesc   DB 0,5
              DW 1,0,0
; Дескриптор команды "Get Configuration Descriptor"
GetConfDesc   DB 80h,6
              DW 200h,0,8
; Дескриптор команды "Set Configuration"
SetConfigur   DB 00h,9
              DW 1,0,0
; Дескриптор команды "Get Port Status"
GetPortStatus DB 0A1h,1
              DW 0,0,1
; Дескриптор команды "Get Device ID"
GetDeviceID   DB 0A1h,0
              DW 0,0,8
ENDS

; Область памяти для хранения дескрипторов передачи
SEGMENT USB_DESCR para public 'DATA'
; Заголовок очереди дескрипторов
QH_Descriptor DD  00000003h   ;единственный заголовок
              DD  00000000h   ;указатель на первый TD
              DD  0,0,0,0,0,0 ;область данных ПО
; Список дескрипторов для одной транзакции
TD_Array      DD  8*64 DUP(?)
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC USB_EpsonStylus_Test
```

```
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Проверить наличие PCI BIDS
        mov     AX,0B101h
        int     1Ah
        jc      @@PCIBIDSNotFound
        cmp     EDX,20494350h
        jne     @@PCIBIDSNotFound
; Вывести текстовые сообщения на экран
        MShowColorText 3,Txt0
        call    GetChar
; Установить режим прямой адресации памяти
        call    Initialization
; Инициализировать дескрипторы USB
        call    InitializeDeascriptors

; ЦИКЛ ПОИСКА ХОСТ-КОНТРОЛЛЕРОВ
        mov     [SearchResult],0
        mov     [USB_HostIndex],0
@@NextHost:
; Найти контроллер USB
        call    FindUSBController
        cmp     [SearchResult],0
        jne     @@NoHost
; Произвести глобальный сброс контроллера
        mov     DX,[USB_BaseAddr]
        mov     AX,100b         ;установить сигнал сброса
        out     DX,AX
; Ожидать не менее 10 мс
        call    Wait05s
; Снять сигнал сброса
        mov     AX,0
        out     DX,AX
; Ожидать не менее 10 мс
        call    Wait05s
; Обнулить счетчик номеров
        mov     [USB_Device_Number],0
; Загрузить указатель на список кадров в регистр
; адреса списка кадров
        mov     DX,[USB_BaseAddr]
        add     DX,6
        mov     AX,0
        out     DX,AX
```

продолжение ⇒

Листинг 8.3 (*продолжение*)

```
        add     DX,2
        mov     EAX,FrameListBaseAddr
        out     DX,EAX
; Активизировать хост-контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,1
        out     DX,AX
; Проверить регистр состояния порта 1
        mov     [USB_PortNum],1
        ; Вычислить адрес регистра состояния порта
        mov     DX,[USB_BaseAddr]
        add     DX,10h
        ; Запомнить адрес регистра состояния порта
        mov     [USB_PortReg],DX
        ; Проверить наличие устройства
        in      AX,DX
        test    AX,000Fh
        jz      @@TestPort2
        ; Присвоить устройству порядковый номер
        call    Enumeration
        ; Получить дескриптор конфигурации
        call    GetConfigurationDescriptor
        ; Устройство является принтером?
        cmp     [word ptr DataBuffer+9+5],0107h
        je      @@PrinterFound
; Проверить регистр состояния порта 2
@@TestPort2:
        mov     [USB_PortNum],2
        ; Вычислить адрес регистра состояния порта
        mov     DX,[USB_BaseAddr]
        add     DX,12h
        ; Запомнить адрес регистра состояния порта
        mov     [USB_PortReg],DX
        ; Проверить наличие устройства
        in      AX,DX
        test    AX,000Fh
        jz      @@ContrStop
        ; Присвоить устройству порядковый номер
        call    Enumeration
        ; Получить дескриптор конфигурации
        call    GetConfigurationDescriptor
        ; Устройство является принтером?
        cmp     [word ptr DataBuffer+9+5],0107h
        je      @@PrinterFound
; Остановить контроллер
@@ContrStop:
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
```

```
        jmp     @@NextHost
@@PrinterFound:

; СКОНФИГУРИРОВАТЬ УСТРОЙСТВО
; Подать команду "Set Configuration"
        mov     SI,offset SetConfigur
        call    Setup_Transaction
; Подать команду "Get Port Status"
        mov     SI,offset GetPortStatus
        call    StatusIN_Transaction
; Проверить состояние принтера
        cmp     [byte ptr DataBuffer],18h
        jne     @@PrinterError

; ИДЕНТИФИЦИРОВАТЬ УСТРОЙСТВО
; Подать команду "Get Device ID"
        mov     SI,offset GetDeviceID
        call    StatusIN_Transaction
; Определить полную длину дескриптора
        mov     AX,[word ptr DataBuffer]
        xchg    AL,AH    ;переставить байты
        mov     [word ptr GetDeviceID+6],AX
; Подать команду "Get Device ID" повторно
        mov     SI,offset GetDeviceID
        call    StatusIN_Transaction
; Проверить тип принтера по идентификатору
        mov     SI,offset DataBuffer
        mov     CX,[SI]  ;загрузить длину строки
        xchg    CL,CH    ;переставить байты
        add     SI,2
        cmp     CX,4
        jbe     @@DeviceNotFound
        sub     CX,4
        ; Цикл поиска слова "EPSON"
        mov     EAX,"OSPE"
@@SearchEPSO:
        cmp     EAX,[SI]
        je      @@SearchStyl
        inc     SI
        loop    @@SearchEPSO
        jmp     @@DeviceNotFound
        ; Цикл поиска слова "Stylus"
        mov     EAX,"lytS"
@@SearchStyl:
        cmp     EAX,[SI]
        je      @@ShowDeviceID
        inc     SI
        loop    @@SearchStyl
        jmp     @@DeviceNotFound
; Показать полученный дескриптор
```

продолжение ⇨

Листинг 8.3 (*продолжение*)

```
@@ShowDeviceID:
        call    ClearScreen
        MShowColorString Txt0
        MShowColorText 2,Txt1
        mov     [ScreenString],19
        mov     CX,[word ptr GetDeviceID+6]
        call    Show_Ident
        call    GetChar

; ПОДГОТОВКА К НАЧАЛУ ПЕЧАТИ ИЗОБРАЖЕНИЯ
; Сбросить триггер данных
        mov     [dword ptr DataTrigger],0
; Выйти из "Packet Mode"
        mov     SI,offset ExitPacketMode
        call    OutCommandToPrn
; Инициализировать принтер
        mov     SI,offset PrnInitialization
        call    OutCommandToPrn
; Включить графический режим печати
        mov     SI,offset SelectGraphicsMode
        call    OutCommandToPrn
; Выбрать монохромный режим
        mov     SI,offset MonochromeSelection
        call    OutCommandToPrn
; Передать команды на принтер
        call    BulkOUT_Transaction

; ОСНОВНОЙ ЦИКЛ (ПО ПЕЧАТАЕМЫМ СТРОКАМ)
        ; Сбросить счетчик строк растра
        mov     [PrintingString],0
        ; Задать начальное значение байта штриховки
        mov     DL,80h
@@P0:
; Задать длину строки 40 байт (320/8)
        mov     SI,offset PrintRasterData
        call    OutCommandToPrn
; Верхнюю и нижнюю строки закрасить полностью
        cmp     [PrintingString],0
        je      @@TopOrBottomLine
        cmp     [PrintingString],319
        jne     @@Shade
@@TopOrBottomLine:
        mov     CX,40
        mov     AL,0FFh ;сплошная линия
; Цикл по печатаемым байтам
@@P1:   call    OutCharToPrn
        loop    @@P1
        jmp     @@LF
; Заштриховать квадрат
```

```
@@Shade:
        ; Левая граница
        mov     AL,DL
        or      AL,80h
        call    OutCharToPrn
        ; Внутренняя часть
        mov     CX,38
        mov     AL,DL
@@P2:   call    OutCharToPrn
        loop    @@P2
        ; Правая граница
        mov     AL,DL
        or      AL,01h
        call    OutCharToPrn
        ; Повернуть байт штриховки влево
        rol     DL,1
; Перейти на следующую строку растра принтера
@@LF:   mov     SI,offset SetRelVertPosition
        call    OutCommandToPrn
; Послать на принтер команду возврата каретки
        mov     AL,0Dh
        call    OutCharToPrn
; Передать данные на принтер
        call    BulkOUT_Transaction
; Перейти на следующую строку экранного изображения
        inc     [PrintingString]
        cmp     [PrintingString],320
        jl      @@P0

; Послать на принтер коды завершения страницы
        mov     AL,0Ch ;перевод формата
        call    OutCharToPrn
; Инициализировать принтер
        mov     SI,offset PrnInitialization
        call    OutCommandToPrn
        call    BulkOUT_Transaction

; Остановить контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h

; Обработка ошибок
@@NoHost:
```

продолжение ➩

Листинг 8.3 (*продолжение*)

```
        cmp     [USB_HostIndex],0
        je      @@HostNotFound
        jmp short @@DeviceNotFound
; Не поддерживается PCI BIOS
@@PCIBIOSNotFound:
        MFatalError NoPCI
; Неверный номер регистра
@@BadRegisterNumber:
        MFatalError BadRg
; Нет ни одного контроллера USB
@@HostNotFound:
        MFatalError NoUSB
; Устройство USB не найдено
@@DeviceNotFound:
        ; Остановить контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
        MFatalError NoPrn
; Принтер не готов к печати
@@PrinterError:
        ; Остановить контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
        MFatalError PrnEr
ENDP USB_EpsonStylus_Test

;*******************************************
;*    ПЕРЕДАТЬ МАССИВ ДАННЫХ УСТРОЙСТВУ USB    *
;* Передаваемые параметры:                     *
;* BULK_DataSize - объем передаваемых данных.  *
;*******************************************
PROC BulkOUT_Transaction near
        pushad
; Загрузить в ESI указатель на массив дескрипторов
        mov     ESI,[Addr_TD_Array]
; Загрузить в EBX указатель на буфер данных
        mov     EBX,[Addr_DataDescr]
; Вычислить количество полных (64-байтных) блоков
        mov     CX,[BULK_DataSize]
        shr     CX,6         ;количество 64-байтных блоков
        cmp     CX,0
        je      @@ShortDataBlock
@@NextDataBlock:
; Сформировать дескриптор данных
        ; Указатель на следующий TD
        mov     EAX,ESI
        add     EAX,32
```

```
        mov     [GS:ESI],EAX
        ; Слово управления
        mov     EAX,[ShDevType] ;тип устройства
        or      EAX,00800000h   ;признак активности
        mov     [GS:ESI+4],EAX
        ; Маркер
        mov     EAX,[ShFuncNum]   ;номер функции
        or      EAX,07E000E1h     ;передать 64 байта
        or      EAX,[DataTrigger] ;триггер данных
        or      EAX,00008000h     ;конечная точка 1
        mov     [GS:ESI+8],EAX
        xor     [dword ptr DataTrigger],80000h
        mov     [GS:ESI+12],EBX   ;буфер данных
        add     EBX,64
        xor     EAX,EAX
        mov     [GS:ESI+16],EAX
        mov     [GS:ESI+20],EAX
        mov     [GS:ESI+24],EAX
        mov     [GS:ESI+28],EAX
        add     ESI,32
        loop    @@NextDataBlock

; Формирование неполного блока
@@ShortDataBlock:
        ; Вычислить размер последнего (неполного) блока
        mov     DX,[BULK_DataSize]
        and     EDX,111111b
        cmp     DX,0              ;размер больше нуля?
        je      @@NoShortBlock
; Сформировать дескриптор данных короткого блока
        mov     [dword ptr GS:ESI],1b ;последний TD
        ; Слово управления
        mov     EAX,[ShDevType] ;тип устройства
        or      EAX,00800000h   ;признак активности
        mov     [GS:ESI+4],EAX
        ; Маркер
        mov     EAX,[ShFuncNum]   ;номер функции
        dec     DX
        shl     EDX,21
        or      EAX,EDX           ;размер блока
        or      EAX,[DataTrigger] ;триггер данных
        or      EAX,0000B000h     ;конечная точка 1
        or      EAX,0E1h
        mov     [GS:ESI+8],EAX
        xor     [dword ptr DataTrigger],80000h
        mov     [GS:ESI+12],EBX
        xor     EAX,EAX
        mov     [GS:ESI+16],EAX
        mov     [GS:ESI+20],EAX
        mov     [GS:ESI+24],EAX
```

продолжение ⇨

Листинг 8.3 (*продолжение*)

```
        mov     [GS:ESI+28],EAX
        jmp short @@Start
; Нет короткого блока, пометить последний полный блок
; как конечный
@@NoShortBlock:
        sub     ESI,32
        mov     [dword ptr GS:ESI],1b ;последний TD
@@Start:
; Установить указатель на список дескрипторов
; (контроллер начинает передачу данных)
        mov     EAX,[Addr_TD_Array]
        mov     ESI,[Addr_QH]
        add     ESI,4
        mov     [GS:ESI],EAX
; Ожидать завершения операции
@@Wait_OpComplete:
        cmp     [dword ptr GS:ESI],1b
        jne     @@Wait_OpComplete
        mov     [BULK_DataSize],0
        popad
        ret
; Переполнен массив дескрипторов
@@TD_Array_Error:
        ; Остановить контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
        MFatalError DsErr
ENDP BulkOUT_Transaction

;*****************************
;* ВЫВЕСТИ СИМВОЛ НА ПРИНТЕР *
;* Параметры:                *
;* AL - код символа.         *
;*****************************
PROC OutCharToPrn near
        push    BX
        cmp     [BULK_DataSize],4096
        jae     @@Data_Buffer_Full
        mov     BX,offset DataBuffer
        add     BX,[BULK_DataSize]
        mov     [BX],AL
        inc     [BULK_DataSize]
        pop     BX
        ret
; Переполнен буфер данных
@@Data_Buffer_Full:
        ; Остановить контроллер
        mov     DX,[USB_BaseAddr]
```

```
        mov     AX,0
        out     DX,AX
        MFatalError BfErr
ENDP OutCharToPrn

;****************************************
;*       ПОСЛАТЬ КОМАНДУ НА ПРИНТЕР     *
;* Параметры:                           *
;* DS:SI - указатель на строку команды. *
;* Первый байт строки содержит количество *
;* байтов команды, посылаемых на принтер. *
;****************************************
PROC OutCommandToPrn near
        pusha
        cld
; Загрузить счетчик байтов команды в CX
        lodsb
        xor     CX,CX
        mov     CL,AL
@@OutNextByte:
        lodsb
        call    OutCharToPrn
        loop    @@OutNextByte
        popa
        ret
ENDP OutCommandToPrn

;**********************************************
;* ОТОБРАЗИТЬ СОДЕРЖИМОЕ ИДЕНТИФИКАТОРА ПРИНТЕРА *
;**********************************************
PROC Show_Ident near
        pusha
        mov     [ScreenString],12
        mov     [ScreenColumn],0
        mov     SI,offset DataBuffer
        mov     CX,[SI]  ;загрузить длину строки
        xchg    CL,CH    ;переставить байты
        add     SI,2
@@NextByte:
        lodsb
        call    ShowASCIIChar
        loop    @@NextByte
        popa
        ret
ENDP Show_Ident
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
```

продолжение ⇒

Листинг 8.3 (*продолжение*)

```
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить процедуры для работы с контроллером USB
include "list8_01.inc"
END
```

В приведенном примере используется набор команд Epson raster, описанный в главе 7 «Принтеры: печать в растровом режиме». Следует отметить, что при подключении принтера через интерфейс USB сразу после завершения процесса конфигурирования устройству требуется подать специальную команду «Exit Packet Mode», предназначенную для переключения принтера из некоего «пакетного режима Epson» (в открытой документации этот режим не описан) в обычный режим работы. Командная ESC-последовательность для выхода из пакетного режима очень длинная:

```
00h, 00h, 00h, 1Bh, 01h, "@EJL", 20h, "1284.4", 0Ah, "@EJL", 20h, 20h, 20h,
20h, 20h, 0Ah
```

Проверка принадлежности принтера к группе моделей EPSON Stylus осуществляется по полученному от принтера идентификатору устройства: строка должна содержать слова «EPSON» и «Stylus».

ВНИМАНИЕ

Программа из листинга 8.3 не предназначена для тестирования принтеров серий C20 и C40, которые используют специфические значения параметров в команде передачи растровой строки.

Работа с мышью через интерфейс USB

Клавиатура и мышь по классификации, принятой для устройств USB, относятся к группе устройств человеко-машинного интерфейса (Human Interface Devices, сокращенно HID) [92].

Клавиатуры с интерфейсом USB до сих пор почти не применяются, так как в среднем стоят дороже стандартных клавиатур и никаких особых преимуществ в работе не дают. Кроме того, могут возникать проблемы, связанные со старым программным обеспечением для MS-DOS и с BIOS SETUP (теоретически во время начальной загрузки BIOS должен работать с клавиатурой USB в режиме эмуляции клавиатуры PS/2, но на практике эта возможность реализуется не всегда).

Ниже мы будем рассматривать мышь в качестве примера устройства, выполняющего передачу данных по прерываниям, так как работа клавиатуры в документации описана очень неполно.

Код класса для устройств, принадлежащих к группе HID, имеет значение 03h. Мышь и клавиатура участвуют в процессе начальной загрузки компьютера, поэтому их относят к подклассу загрузочных устройств (Boot Devices), который обозначается кодом 01h. Код протокола для клавиатуры имеет значение 01h, а для мыши — значение 02h.

Пакет данных о текущем состоянии устройства HID и выполняемых с ним операциях именуется в документации рапортом (report). Мышь передает хосту рапорты в режиме передачи по прерываниям.

Поскольку мышь является загрузочным устройством, начальный участок рапорта стандартизирован:

- байт 0 содержит информацию о состоянии клавиш мыши;
- байт 1 передает значение перемещения по оси X;
- байт 2 передает значение перемещения по оси Y.

Назначение остальных байтов рапорта мыши определяется изготовителем (для так называемых трехкоординатных устройств координата Z обычно передается в байте 3).

ПРИМЕЧАНИЕ

Значение перемещения передается в виде двоичного числа со знаком (при определении знака предполагается, что ось X направлена слева направо, ось Y — сверху вниз).

Структура байта 0 стандартизирована не полностью:

- бит 0 — состояние клавиши 1 (0 — отпущена, 1 — нажата);
- бит 1 — состояние клавиши 2 (0 — отпущена, 1 — нажата);
- бит 2 — состояние клавиши 3 (0 — отпущена, 1 — нажата);
- биты 3–7 используются по усмотрению изготовителя устройства.

ПРИМЕЧАНИЕ

Значение бита 0 соответствует состоянию левой клавише мыши.

Размер рапорта определяется изготовителем, но не может быть меньше трех байт. Получить размер рапорта в байтах можно из поля максимального размера пакета в дескрипторе конечной точки.

Если включен режим эмуляции стандартного периферийного оборудования (мыши и клавиатуры PS/2), BIOS обрабатывает только

стандартную часть рапорта мыши, а остальные данные отбрасывает. По этому же принципу можно строить простые универсальные программы для работы с изделиями разных изготовителей на аппаратном уровне, то есть принимать рапорт целиком, но обрабатывать только первые три байта.

В листинге 8.4 приведена программа USB_Mouse, которая производит поиск мыши по портам хост-контроллера, а затем выводит курсор мыши на экран; программа контролирует состояние левой кнопки мыши и завершает свою работу при нажатии на нее. Программа использует следующие процедуры:

- процедура IntEndpointDescriptor позволяет определить номер конечной точки, осуществляющей передачу данных от мыши, и размер передаваемого блока данных;
- процедура InterruptIN_Transaction осуществляет прием данных от мыши в режиме передачи по прерываниям;
- процедура ShowNewMouseCursorPosition отображает курсор мыши на экране монитора путем инверсии байта атрибута символа.

Кроме того, программа USB_Mouse использует универсальные процедуры ввода-вывода из листинга 1.2, процедуру переключения в линейный режим адресации из листинга 2.1 и набор процедур для работы с контроллером и устройствами USB из листинга 8.1.

Листинг 8.4. Работа с мышью через интерфейс USB

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Физический адрес области памяти для списка кадров USB
FrameListBaseAddr equ 200000h
; Параметры экрана в текстовом режиме
ScreenLength equ 80  ;количество символов в строке
ScreenHeigth equ 25  ;количество строк на экране

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Старое значение фона символа
OldCharBackground DB 0Fh
; Текущее состояние кнопок
```

```
ButtonsStatus   DB 0
; Текущие координаты курсора мыши
XCoordinate     DW 0
YCoordinate     DW 0
; Предыдущая позиция курсора мыши
OldXCoordinate  DW 0
OldYCoordinate  DW 0
; Текстовые сообщения
Txt0 DB LIGHTBLUE,0,25,"ПОИСК И ТЕСТИРОВАНИЕ МЫШИ USB",0
     DB LIGHTMAGENTA,12,9,"Отображение курсора "
     DB "осуществляется инверсией атрибута символа",0
     DB YELLOW,24,21
     DB "Для выхода нажмите левую клавишу мыши",0
Txt1 DB 2,24,"Порядковый номер контроллера:",0
     DB 4,8,"Базовый адрес набора регистров:",0
     DB 5,8,"Номер используемого прерывания:",0
     DB 7,23,"Регистр команды:",0
     DB 8,21,"Регистр состояния:",0
     DB 9,7,"Регистр управления прерываниями:",0
     DB 10,27,"Номер кадра:",0
     DB 11,11,"Базовый адрес списка кадров:",0
     DB 12,14,"Модификация начала кадра:",0
     DB 13,13,"Регистр состояния порта 1:",0
     DB 14,13,"Регистр состояния порта 2:",0
     DB 16,17,"Адрес активного порта:",0
AnyK DB YELLOW,24,29,"Нажмите любую клавишу",0
; Сообщения об ошибках
NoMouse DB 12,31,"Мышь не обнаружена",0

; ДЕСКРИПТОРЫ КОМАНД
; Дескриптор команды "Get Device Descriptor"
GetDevDesc  DB 80h,6
            DW 100h,0,8
; Дескриптор команды "Set Address"
SetAddrDesc DB 0,5
            DW 0,0,0
; Дескриптор команды "Get Configuration Descriptor"
GetConfDesc DB 80h,6
            DW 200h,0,8
; Дескриптор команды "Set Configuration"
SetConfigur DB 00h,9
            DW 1,0,0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

; Область памяти для хранения дескрипторов передачи
SEGMENT USB_DESCR para public 'DATA'
```

продолжение ⇨

Листинг 8.4 (*продолжение*)

```
; Заголовок очереди дескрипторов
QH_Descriptor DD  00000003h   ;единственный заголовок
              DD  00000000h   ;указатель на первый TD
              DD  0,0,0,0,0,0 ;область данных ПО
; Список дескрипторов для одной транзакции
TD_Array      DD  8*16 DUP(0)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC USB_Mouse
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Проверить наличие PCI BIOS
        mov     AX,0B101h
        int     1Ah
        jc      @@PCIBIOSNotFound
        cmp     EDX,20494350h
        jne     @@PCIBIOSNotFound
; Установить режим прямой адресации памяти
        call    Initialization
; Инициализировать дескрипторы USB
        call    InitializeDeascriptors

; ЦИКЛ ПОИСКА ХОСТ-КОНТРОЛЛЕРОВ
        mov     [SearchResult],0
        mov     [USB_HostIndex],0
@@NextHost:
; Найти контроллер USB
        call    FindUSBController
        cmp     [SearchResult],0
        jne     @@NoHost
; Произвести глобальный сброс контроллера
        mov     DX,[USB_BaseAddr]
        mov     AX,100b         ;установить сигнал сброса
        out     DX,AX
; Ожидать не менее 10 мс
        call    Wait05s
; Снять сигнал сброса
```

```
        mov     AX,0
        out     DX,AX
; Ждать не менее 10 мс
        call    Wait05s
; Обнулить счетчик номеров
        mov     [USB_Device_Number],0
; Загрузить указатель на список кадров в регистр
; адреса списка кадров
        mov     DX,[USB_BaseAddr]
        add     DX,6
        mov     AX,0
        out     DX,AX
        add     DX,2
        mov     EAX,FrameListBaseAddr
        out     DX,EAX
; Активизировать хост-контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,1
        out     DX,AX
; Проверить регистр состояния порта 1
        mov     [USB_PortNum],1
        ; Вычислить адрес регистра состояния порта
        mov     DX,[USB_BaseAddr]
        add     DX,10h
        ; Запомнить адрес регистра состояния порта
        mov     [USB_PortReg],DX
        ; Проверить наличие устройства
        in      AX,DX
        test    AX,000Fh
        jz      @@TestPort2
        ; Присвоить устройству порядковый номер
        call    Enumeration
        ; Получить дескриптор конфигурации
        call    GetConfigurationDescriptor
        ; Устройство является мышью?
        cmp     [byte ptr DataBuffer+9+5],03h
        jne     @@TestPort2
        cmp     [byte ptr DataBuffer+9+7],02h
        je      @@MouseFound
; Проверить регистр состояния порта 2
@@TestPort2:
        mov     [USB_PortNum],2
        ; Вычислить адрес регистра состояния порта
        mov     DX,[USB_BaseAddr]
        add     DX,12h
        ; Запомнить адрес регистра состояния порта
        mov     [USB_PortReg],DX
        ; Проверить наличие устройства
        in      AX,DX
        test    AX,000Fh
```

продолжение

Листинг 8.4 (*продолжение*)

```
        jz      @@ContrStop
        ; Присвоить устройству порядковый номер
        call    Enumeration
        ; Получить дескриптор конфигурации
        call    GetConfigurationDescriptor
        cmp     [byte ptr DataBuffer+9+5],03h
        jne     @@TestPort2
        cmp     [byte ptr DataBuffer+9+7],02h
        je      @@MouseFound
; Остановить контроллер
@@ContrStop:
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
        jmp     @@NextHost

@@MouseFound:
; Вывести текстовые сообщения на экран
        MShowColorText 3,Txt0
; СКОНФИГУРИРОВАТЬ УСТРОЙСТВО
; Подать команду "Set Configuration"
        mov     SI,offset SetConfigur
        call    Setup_Transaction
; Определить адрес конечной точки и размер пакета
        call    IntEndpointDescriptor

; Отобразить курсор мыши первый раз
        call    ShowNewMouseCursorPosition
; Сбросить триггер данных
        mov     [dword ptr DataTrigger],0
@@NextInterrupt:
; Принять от мыши пакет данных
        call    InterruptIN_Transaction
; Прибавить перемещение по X к координате X
        mov     AL,[DataBuffer+1]
        cbw
        add     AX,[XCoordinate]
        js      @@X1
        cmp     AX,ScreenLength
        jb      @@X2
        mov     AX,ScreenLength-1
        jmp     @@X2
@@X1:   xor     AX,AX
@@X2:   mov     [XCoordinate],AX
; Прибавить перемещение по Y к координате Y
        mov     AL,[DataBuffer+2]
        cbw
        add     AX,[YCoordinate]
        js      @@Y1
```

```
        cmp     AX,ScreenHeigth
        jb      @@Y2
        mov     AX,ScreenHeigth-1
        jmp     @@Y2
@@Y1:   xor     AX,AX
@@Y2:   mov     [YCoordinate],AX
; Показать курсор в новой позиции
        call    ShowNewMouseCursorPosition
; Проверить состояние левой кнопки
        test    [DataBuffer],00000001b
        jz      @@NextInterrupt

; Остановить контроллер
        mov     DX,[USB_BaseAddr]
        mov     AX,0
        out     DX,AX
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h

; Обработка ошибок
@@NoHost:
        cmp     [USB_HostIndex],0
        je      @@HostNotFound
        jmp short @@MouseNotFound
; Не поддерживается PCI BIOS
@@PCIBIOSNotFound:
        MFatalError NoPCI
; Неверный номер регистра
@@BadRegisterNumber:
        MFatalError BadRg
; Нет ни одного контроллера USB
@@HostNotFound:
        MFatalError NoUSB
; Мышь USB не найдена
@@MouseNotFound:
        MFatalError NoMouse
ENDP USB_Mouse

;*******************************************
;* ОТОБРАЖЕНИЕ КУРСОРА МЫШИ ПУТЕМ ИНВЕРСИИ *
;*   АТРИБУТА СИМВОЛА В ПОЗИЦИИ КУРСОРА    *
;*******************************************
PROC ShowNewMouseCursorPosition NEAR
        pusha
        push    ES
        ; Настроить ES на видеопамять
```

продолжение ⇨

Листинг 8.4 (*продолжение*)

```
        mov     AX,0B800h
        mov     ES,AX
        ; Вычислить старую координату курсора
        mov     AX,[OldYCoordinate]
        mov     DX,160
        mul     DX
        add     AX,[OldXCoordinate]
        add     AX,[OldXCoordinate]
        inc     AX
        mov     DI,AX
        ; Восстановить атрибут символа
        mov     AL,[OldCharBackground]
        mov     [ES:DI],AL
        ; Вычислить новую координату курсора
        mov     AX,[YCoordinate]
        mov     DX,160
        mul     DX
        add     AX,[XCoordinate]
        add     AX,[XCoordinate]
        inc     AX
        mov     DI,AX
        ; Сохранить атрибут символа
        mov     AL,[ES:DI]
        mov     [OldCharBackground],AL
        ; Инвертировать атрибут
        xor     [byte ptr ES:DI],1111111b
        ; Запомнить координаты символа
        mov     AX,[XCoordinate]
        mov     [OldXCoordinate],AX
        mov     AX,[YCoordinate]
        mov     [OldYCoordinate],AX
        pop     ES
        popa
        ret
ENDP ShowNewMouseCursorPosition

;*****************************************
;*        ПРИНЯТЬ ПАКЕТ ПО ПРЕРЫВАНИЮ        *
;* Передаваемые параметры:                  *
;* INT_DataSize - объем принимаемых данных. *
;*****************************************
PROC InterruptIN_Transaction near
        pushad
; Загрузить в ESI указатель на массив дескрипторов
        mov     ESI,[Addr_TD_Array]
; Загрузить в EBX указатель на буфер данных
        mov     EBX,[Addr_DataDescr]

; Сформировать дескриптор данных
```

```
        ; Указатель на следующий TD
        mov     [dword ptr GS:ESI],1b ;последний TD
        ; Слово управления
        mov     EAX,[ShDevType]  ;тип устройства
        or      EAX,00800000h    ;признак активности
        mov     [GS:ESI+4],EAX
        ; Маркер
        mov     EAX,69h          ;прием данных
        or      EAX,[ShFuncNum]  ;номер функции
        or      EAX,[ShEndpNum]  ;конечная точка
        or      EAX,[DataTrigger]
        or      EAX,[ShPackSize] ;размер блока
        mov     [GS:ESI+8],EAX
        ; Переключить триггер данных
        xor     [dword ptr DataTrigger],80000h
        mov     [GS:ESI+12],EBX ;буфер данных
        xor     EAX,EAX
        mov     [GS:ESI+16],EAX
        mov     [GS:ESI+20],EAX
        mov     [GS:ESI+24],EAX
        mov     [GS:ESI+28],EAX

; Установить указатель на список дескрипторов
; (контроллер начинает передачу данных)
        mov     EAX,[Addr_TD_Array]
        mov     ESI,[Addr_QH]
        add     ESI,4
        mov     [GS:ESI],EAX
; Ожидать завершения операции
@@Wait_OpComplete:
        cmp     [dword ptr GS:ESI],1b
        jne     @@Wait_OpComplete
        popad
        ret
ENDP InterruptIN_Transaction

;****************************************
;* ОПРЕДЕЛИТЬ МАКСИМАЛЬНЫЙ РАЗМЕР ПАКЕТА *
;*    ДЛЯ ИСПОЛЬЗУЕМОЙ КОНЕЧНОЙ ТОЧКИ    *
;****************************************
PROC IntEndpointDescriptor near
        pusha
; Поиск дескриптора конечной точки в списке дескрипторов
        mov     BX,0 ;счетчик байтов
@@NextDescriptor:
        cmp     [word ptr DataBuffer+BX],0507h
        je      @@Endpoint
@@NextDescOffset:
        ; Вычислить смещение следующего дескриптора
        add     BL,[DataBuffer+BX]
```

продолжение ⇨

Листинг 8.4 (*продолжение*)

```
        adc     BH,0
        ; Проверка на превышение длины массива
        cmp     BX,[word ptr DataBuffer+2]
        jb      @@NextDescriptor
        MFatalError NoDev
@@Endpoint:
; Точка передачи по прерываниям?
        test    [DataBuffer+BX+2],10000000b
        jz      @@NextDescOffset
        cmp     [DataBuffer+BX+3],3
        jne     @@NextDescDffset
; Запомнить адрес конечной точки
        xor     EAX,EAX
        mov     AL,[DataBuffer+BX+2]
        and     AL,00001111b
        shl     EAX,15
        mov     [ShEndpNum],EAX
; Запомнить размер пакета
        xor     EAX,EAX
        mov     AX,[word ptr DataBuffer+BX+4]
        dec     AX
        shl     EAX,21
        mov     [ShPackSize],EAX
        popa
        ret
ENDP IntEndpointDescriptor
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить подпрограмму, переводящую сегментный
; регистр GS в режим линейной адресации
include "list2_01.inc"
; Подключить процедуры для работы с контроллером USB
include "listB_01.inc"

END
```

ПРИМЕЧАНИЕ

Программа USB_Mouse осуществляет поиск мыши только непосредственно по портам хост-контроллера, поэтому перед запуском теста нужно подсоединить мышь к одному из USB-портов системного блока.

Глава 9 NE2000-совместимые сетевые адаптеры

NE2000 – наименование изделия фирмы Novell. Адаптеры NE2000 предназначались для работы в сети Ethernet и в свое время были настолько популярными, что стали фактическим стандартом. Адаптеры, спроектированные другими фирмами на основе конструкции, предложенной фирмой Novell, получили название NE2000-совместимых.

NE2000-совместимые адаптеры позволяют передавать данные со скоростью 10 Мбит/с по стандартному коаксиальному кабалю или по витой паре проводов. До сих пор группа NE2000-совместимых адаптеров является единственным примером общеотраслевого стандарта на архитектуру адаптеров Ethernet: для следующих поколений адаптеров (Fast Ethernet со скоростью передачи 100 Мбит/с и Gigabit Ethernet со скоростью передачи 1 Гбит/с) каждый разработчик стал использовать свою собственную архитектуру.

Несмотря на то, что со времени разработки адаптера NE2000 скорость передачи сети Ethernet увеличилась на два порядка (с 10 Мбит/с до 1 Гбит/с), совместимые с ним изделия до сих пор продолжают выпускаться: низкая скорость работы компенсируется невысокой ценой и совместимостью с любыми операционными системами и старым программным обеспечением. Кроме того, сети, построенные на основе коаксиальных кабелей, не поддерживают скорость передачи выше 10 Мбит/с, и использование в таких сетях более современных сетевых адаптеров не дает никаких преимуществ.

Низкая цена (менее $10) и наличие общего стандарта делают NE2000-совместимые адаптеры привлекательными для использования в целях обучения, особенно при проведении лабораторных работ, связанных с исследованием работы сети Ethernet на аппаратном уровне. Определенную пользу в этом случае можно получить даже от невысокой

скорости работы таких адаптеров — для наблюдения за процессом передачи можно использовать «учебные» осциллографы, частотные характеристики которых обычно оставляют желать лучшего.

Регистры NE2000-совместимого адаптера

NE2000-совместимые адаптеры выпускаются в двух вариантах исполнения: для шины ISA и для шины PCI. Шина ISA практически вышла из употребления, поэтому ниже мы будем рассматривать только вариант, предназначенный для шины PCI.

Регистровые страницы

Доступные для пользователя регистры NE2000-совместимого адаптера Ethernet распределены между тремя страницами, которые разделяют между собой один и тот же участок пространства вводавывода [41–43].

Каждая страница содержит по 16 регистров. Регистры страницы нулевой перечислены в табл. 9.1, регистры первой страницы — в табл. 9.2, регистры второй страницы — в табл. 9.3.

Таблица 9.1. Регистры нулевой страницы NE2000-совместимого адаптера Ethernet

Адрес	Считываемый регистр	Записываемый регистр
00h	Регистр команды (CR)	Регистр команды (CR)
01h	Младший байт текущего адреса локального канала DMA (CLDA0)	Регистр номера начальной страницы кольцевого буфера (PSTART)
02h	Старший байт текущего адреса локального канала DMA (CLDA1)	Регистр номера конечной страницы кольцевого буфера (PSTOP)
03h	Указатель границы (BNRY)	Указатель границы (BNRY)
04h	Регистр состояния передатчика (TSR)	Номер начальной страницы области памяти передатчика (TPSR)
05h	Счетчик коллизий (NCR)	Младший байт счетчика передаваемых байтов (TBCR0)
06h	Регистр очереди данных (FIFO)	Старший байт счетчика передаваемых байтов (TBCR1)

Адрес	Считываемый регистр	Записываемый регистр
07h	Регистр статуса прерывания (ISR)	Регистр статуса прерывания (ISR)
08h	Младший байт текущего адреса DMA для операций внешнего доступа (CRDA0)	Младший байт начального адреса для операций внешнего доступа (RSAR0)
09h	Старший байт текущего адреса DMA для операций внешнего доступа (CRDA1)	Старший байт начального адреса для операций внешнего доступа (RSAR1)
0Ah	Зарезервирован	Младший байт счетчика байтов для операций внешнего доступа (RBCR0)
0Bh	Зарезервирован	Старший байт счетчика байтов для операций внешнего доступа (RBCR1)
0Ch	Регистр состояния приемника (RSR)	Регистр управления приемником (RCR)
0Dh	Счетчик ошибок выравнивания (CNTR0)	Регистр управления передатчиком (TCR)
0Eh	Счетчик ошибок CRC (CNTR1)	Регистр управления форматом данных (DCR)
0Fh	Счетчик потерянных пакетов (CNTR2)	Регистр маскирования прерываний (IMR)

Таблица 9.2. Регистры первой страницы NE2000-совместимого адаптера Ethernet

Адрес	Регистр
00h	Регистр команды (CR)
01h	Байт 0 физического адреса (PAR0)
02h	Байт 1 физического адреса (PAR1)
03h	Байт 2 физического адреса (PAR2)
04h	Байт 3 физического адреса (PAR3)
05h	Байт 4 физического адреса (PAR4)
06h	Байт 5 физического адреса (PAR5)
07h	Регистр номера текущей страницы (CURR)
08h	Байт 0 группового адреса (MAR0)
09h	Байт 1 группового адреса (MAR1)
0Ah	Байт 2 группового адреса (MAR2)
0Bh	Байт 3 группового адреса (MAR3)
0Ch	Байт 4 группового адреса (MAR4)

продолжение ⇒

Таблица 9.2 (*продолжение*)

Адрес	Регистр
0Dh	Байт 5 группового адреса (MAR5)
0Eh	Байт 6 группового адреса (MAR6)
0Fh	Байт 7 группового адреса (MAR7)

Таблица 9.3. Регистры второй страницы NE2000-совместимого адаптера Ethernet

Адрес	Считываемый регистр	Записываемый регистр
00h	Регистр команды (CR)	Регистр команды (CR)
01h	Регистр номера начальной страницы кольцевого буфера (PSTART)	Младший байт текущего адреса локального канала DMA (CLDA0)
02h	Регистр номера конечной страницы кольцевого буфера (PSTOP)	Старший байт текущего адреса локального канала DMA (CLDA1)
03h	Указатель на следующий пакет для внешнего доступа	Указатель на следующий пакет для внешнего доступа
04h	Номер начальной страницы области памяти передатчика (TPSR)	Зарезервирован
05h	Внутренний указатель на следующий пакет	Внутренний указатель на следующий пакет
06h	Старший байт счетчика адресов	Старший байт счетчика адресов
07h	Младший байт счетчика адресов	Младший байт счетчика адресов
08h	Зарезервирован	Зарезервирован
09h	Зарезервирован	Зарезервирован
0Ah	Зарезервирован	Зарезервирован
0Bh	Зарезервирован	Зарезервирован
0Ch	Регистр управления приемником (RCR)	Зарезервирован
0Dh	Регистр управления передатчиком (TCR)	Зарезервирован
0Eh	Регистр управления форматом данных (DCR)	Зарезервирован
0Fh	Регистр маскирования прерываний (IMR)	Зарезервирован

Внутренние регистры адаптера

Регистр команд (Command register, сокращенно **CR**), доступный с любой страницы, размещается по базовому адресу (без смещения). Он предназначен для выбора страницы, запуска процесса передачи пакета, а также блокировки и разблокировки внешних DMA-операций.

7	6	5	4	3	2	1	0
PS1	PS0	RD2	RD1	RD0	TXP	STA	STP

Рис. 9.1. Структура регистра команд CR

Структура регистра команд показана на рис. 9.1. Разряды регистра имеют следующее назначение:

- бит 0 (STP) предназначен для программного сброса адаптера: после установки в 1 данного разряда работа адаптера приостанавливается (если в момент установки разряда выполняется операция приема или передачи пакета, останов производится после полного завершения этой операции; выход из состояния останова осуществляется путем сброса данного разряда);
- бит 1 (STA) предназначен для активизации адаптера (установка данного бита в 1 позволяет активизировать работу адаптера после включения питания, программного сброса или обнаружения сбоя);
- бит 2 (TXP) обеспечивает запуск операции передачи пакета (установка данного бита в 1 инициирует операцию передачи пакета; после завершения операции передачи бит TXP сбрасывается в 0);
- биты 3–5 (RD0, RD1, RD2) содержат код внешней DMA-операции (см. табл. 9.4);
- биты 6–7 (PS0, PS1) служат для выбора регистровой страницы адаптера (см. табл. 9.5).

Таблица 9.4. Значение битов RD0–RD2 регистра команд

Код команды			**Выполняемая операция**
RD2	**RD1**	**RD0**	
0	0	0	Запрещенная комбинация
0	0	1	Передача данных от адаптера в память компьютера (Remote Read)

продолжение ⇗

Таблица 9.4 (*продолжение*)

Код команды			Выполняемая операция
RD2	RD1	RD0	
0	1	0	Запись данных из памяти компьютера в память адаптера (Remote Write)
0	1	1	Передача пакета
1	X	X	Аварийное прекращение внешней DMA-операции

Таблица 9.5. Значение битов PS0 и PS1 регистра команд

Код страницы		Номер страницы
PS1	PS0	
0	0	Регистровая страница 0
0	1	Регистровая страница 1
1	0	Регистровая страница 2
1	1	Резервная или тестовая страница (пользователям недоступна)

Регистр статуса прерывания (Interrupt Status Register, сокращенно **ISR**) позволяет центральному процессору компьютера определить причину прерывания.

7	6	5	4	3	2	1	0
RST	RDC	CNT	OVW	TXE	RXE	PTX	PRX

Рис. 9.2. Структура регистра статуса прерывания ISR

Структура регистра ISR показана на рис. 9.2. Разряды регистра имеют следующее назначение:

- бит 0 (PRX) — признак поступления пакета данных (устанавливается в 1 после завершения приема пакета, если пакет принят без ошибок);
- бит 1 (PTX) — признак завершения передачи пакета данных (устанавливается в 1 после завершения передачи пакета, если пакет передан без ошибок);
- бит 2 (RXE) — признак того, что пакет принят с одной или несколькими ошибками (устанавливается в 1 при обнаружении ошибки по CRC, ошибки выравнивания кадра, при переполнении FIFO или потере пакета);

- бит 3 (TXE) — признак того, что при передаче пакета произошла ошибка (устанавливается в 1, если количество коллизий превысило допустимое значение или буфер FIFO не был полностью загружен);
- бит 4 (OVW) — признак переполнения кольцевого буфера приемника (устанавливается в 1, если при выполнении внутренней DMA-операции был перейден ограничитель, указывающий на начало первого занятого блока буфера — Boundary Pointer);
- бит 5 (CNT) — признак переполнения счетчика ошибок (устанавливается в 1 при переполнении любого из трех счетчиков ошибок — счетчика ошибок выравнивания, счетчика ошибок CRC или счетчика потерянных пакетов);
- бит 6 (RDC) — признак завершения внешней DMA-операции (устанавливается в 1, когда операция завершена);
- бит 7 (RST) — признак сброса адаптера (данный бит *не генерирует прерывание* и служит только индикатором состояния адаптера; бит RST устанавливается в 1 после сброса адаптера и сбрасывается в 0 после подачи команды запуска, то есть после установки бита STA в регистре команд).

ВНИМАНИЕ

Сигнал прерывания определенного типа сбрасывается путем записи *единицы* в соответствующий разряд регистра ISR. После включения питания нужно сбросить регистр ISR, записав в него код FFh (то есть единицы во все разряды).

Регистр маскирования прерываний (Interrupt Mask Register, сокращенно **IMR**) позволяет избирательно блокировать (запрещать) некоторые типы запросов прерывания от сетевого адаптера.

7	6	5	4	3	2	1	0
—	RDCE	CNTE	OVWE	TXEE	RXEE	PTXE	PRXE

Рис. 9.3. Структура регистра маскирования прерываний IMR

Структура регистра IMR показана на рис. 9.3. Разряды регистра имеют следующее назначение:

- бит 0 (PRXE) — управление сигналом завершения приема пакета (0 — прерывание запрещено, 1 — прерывание разрешено);
- бит 1 (PTXE) — управление сигналом завершения передачи пакета (0 — прерывание запрещено, 1 — прерывание разрешено);

- бит 2 (RXEE) — управление сигналом об обнаружении ошибок при приеме пакета (0 — прерывание запрещено, 1 — прерывание разрешено);
- бит 3 (TXEE) — управление сигналом об обнаружении ошибок при передаче пакета (0 — прерывание запрещено, 1 — прерывание разрешено);
- бит 4 (OVWE) — управление сигналом переполнения кольцевого буфера (0 — прерывание запрещено, 1 — прерывание разрешено);
- бит 5 (CNTE) — управление сигналом переполнения счетчика ошибок (0 — прерывание запрещено, 1 — прерывание разрешено);
- бит 6 (RDCE) — управление сигналом завершения внешней DMA-операции (0 — прерывание запрещено, 1 — прерывание разрешено);
- бит 7 — зарезервирован.

Таким образом, каждый разряд регистра IMR (за исключением седьмого — прерывание по сбросу адаптера не вырабатывается и, соответственно, не может быть замаскировано) соответствует разряду регистра ISR с тем же порядковым номером.

Сигнал прерывания вырабатывается адаптером, если в регистре ISR установлен хотя бы один не замаскированный регистром IMR бит, и остается активным до тех пор, пока незамаскированные разряды регистра ISR не будут сброшены (путем записи в них единиц).

После включения питания компьютера разряды регистра IMR устанавливаются в 0: все сигналы, кроме сигнала сброса, замаскированы.

Регистр управления форматом данных (Data Configuration Register, сокращенно **DCR**), предназначен для выбора разрядности интерфейса памяти (8- или 16-разрядный), порядка размещения байтов в 16-разрядных словах и установки порога FIFO.

7	6	5	4	3	2	1	0
—	FT1	FT0	ARM	LS	LAS	BOS	WTS

Рис. 9.4. Структура регистра управления форматом данных DCR

Структура регистра DCR показана на рис. 9.4. Разряды регистра имеют следующее назначение:

- бит 0 (WTS) — выбор ширины шины данных для внешних и внутренних DMA-операций (0 — 8 разрядов, 1 — 16 разрядов);

- бит 1 (BOS) — выбор порядка байтов в словах (0 — порядок «младший байт — первый», принятый для процессоров 80x86 фирмы Intel; 1 — порядок «старший байт — первый», принятый для процессоров фирмы Motorola);
- бит 2 (LAS) — выбор режима адресации памяти компьютера (0 — используется спаренный 16-разрядный канал DMA, 1 — используется 32-разрядный канал DMA);
- бит 3 (LS) — выбор режима работы адаптера (0 — режим диагностики, 1 — рабочий режим);
- бит 4 (ARM) — управление режимом автоинициализации (0 — команда передачи пакета не выполняется, все пакеты удаляются из кольцевого буфера под управлением центрального процессора компьютера; 1 — допускается выполнение команды передачи пакета, производится автоинициализация внешней DMA-операции для удаления пакетов из кольцевого буфера);
- биты 5–6 (FT0 и FT1) определяют «предел» FIFO, то есть число байтов или слов FIFO, реально используемых при приеме и передаче данных (табл. 9.6);
- бит 7 зарезервирован.

Таблица 9.6. Зависимость значения предела FIFO от заданной ширины шины данных DMA и значения битов FT0 и FT1 регистра DCR

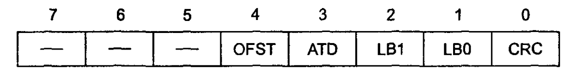
Код предела FIFO		Ширина шины данных DMA	
FT1	**FT0**	**16 разрядов**	**8 разрядов**
0	0	1 слово	2 байта
0	1	2 слова	4 байта
1	0	4 слова	8 байт
1	1	6 слов	12 байт

В современных NE2000-совместимых сетевых адаптерах значение некоторых разрядов регистра DCR жестко (на аппаратном уровне) зафиксировано:

- бит BOS всегда содержит значение 0 (используется порядок байтов процессоров серии 80x86);
- бит LAS всегда содержит значение 0 (используется 16-разрядный режим DMA);
- бит ARM всегда содержит значение 0 (автоинициализация кольцевого буфера и использование команды передачи пакета запрещены);

- бит FT1 всегда содержит значение 1, а бит FT0 — значение 0 (что соответствует пределу FIFO в 4 слова).

Регистр управления передатчиком (Transmit Configuration Register, сокращенно **TCR**) задает параметры работы передающего блока сетевого адаптера.



7	6	5	4	3	2	1	0
—	—	—	OFST	ATD	LB1	LB0	CRC

Рис. 9.5. Структура регистра управления передатчиком TCR

Структура регистра TCR показана на рис. 9.5. Разряды регистра имеют следующее назначение:

- бит 0 (CRC) служит для управления схемой генерации контрольного кода (0 — обычный режим работы, код CRC добавляется в конец пакета при передаче; 1 — самотестирование адаптера, код CRC не формируется);
- биты 1–2 (LB0 и LB1) содержат код режима работы передатчика (см. табл. 9.7);
- бит 3 (ATD) служит для управления опцией внешней блокировки передатчика (0 — обычный режим работы, внешняя блокировка передатчика запрещена; 1 — разрешена внешняя блокировка передатчика);
- бит 4 (OFST) служит для выбора алгоритма обработки коллизий (0 — стандартный алгоритм, 1 — модифицированный алгоритм);
- биты 5–7 не используются (зарезервированы).

Рассмотрим более подробно разряды ATD и OFST. Установка в 1 разряда ATD позволяет другим станциям сети блокировать и разблокировать работу передатчика путем передачи пакета с определенным групповым адресом (поступление пакета с групповым адресом, хеширующим бит 62, вызывает отключение передатчика; пакет с групповым адресом, хеширующим бит 63, позволяет включить передатчик).

Установка в 1 разряда OFST приводит к тому, что адаптер начинает использовать модифицированный («смещенный») алгоритм обработки коллизий: после первой коллизии переходит в режим с самым низким приоритетом (с максимальной задержкой передачи), остается в нем после двух следующих коллизий, а затем переходит к обычному алгоритму обработки коллизий.

Таблица 9.6. Значение кода режима работы передатчика

Код режима		Состояние сигнала LPBK	Номер режима	Режим работы
LB1	LB0			
0	0	0	0	Обычный
0	1	0	1	Внутренняя диагностика
1	0	1	2	Внешняя диагностика
1	1	0	3	Внешняя диагностика

Регистр состояния передатчика (Transmit Status Register, сокращенно **TSR**) отражает текущее состояние передатчика.

7	6	5	4	3	2	1	0
OWC	CDH	FU	CRS	ABT	COL	—	PTX

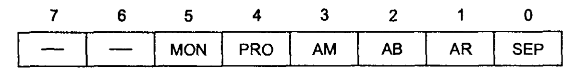
Рис. 9.6. Структура регистра состояния передатчика TSR

Структура регистра TSR показана на рис. 9.6. Разряды регистра имеют следующее назначение:

- бит 0 (PTX) — признак успешной передачи пакета (устанавливается в 1, если пакет передан без ошибок);
- бит 1 — зарезервирован;
- бит 2 (COL) — признак наличия коллизий (устанавливается в 1, если в процессе передачи пакета имела место хотя бы одна коллизия);
- бит 3 (ABT) — признак аварийного завершения передачи пакета (устанавливается в 1, если передача пакета прервана аварийно после 16 коллизий на линии связи);
- бит 4 (CRS) — признак потери несущей частоты (устанавливается в 1, если при передаче произошла потеря несущей частоты; передача пакета, однако, в этом случае не прекращается);
- бит 5 (FU) — признак переполнения FIFO (устанавливается в 1, если в процессе передачи пакета произошло переполнение буфера FIFO; передача пакета в этом случае немедленно прекращается);
- бит 6 (CDH) — признак сбоя передатчика при формировании сигнала коллизии (устанавливается в 1, если в течение заданного стандартом Ethernet интервала времени сигнал коллизии на линии не сформирован);

- бит 7 (OWC) — признак запаздывания сигнала коллизии (устанавливается в 1, если сигнал коллизии на линии сформирован с опозданием, превышающим стандартную величину, равную 51,2 мкс; в этом случае, как и при обнаружении обычной коллизии, производится повторная передача пакета).

Регистр управления приемником (Receive Configuration Register, сокращенно **RCR**) задает параметры работы блока приемника сетевого адаптера.



7	6	5	4	3	2	1	0
—	—	MON	PRO	AM	AB	AR	SEP

Рис. 9.7. Структура регистра управления приемником RCR

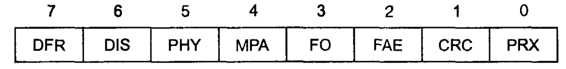
Структура регистра RCR показана на рис. 9.7. Разряды регистра имеют следующее назначение:

- бит 0 (SEP) управляет режимом обработки пакетов, принятых с ошибками (0 — пакеты с ошибками не сохраняются; 1 — сохраняются пакеты с ошибками CRC и выравнивания кадров);
- бит 1 (AR) служит для управления обработкой «коротких» пакетов с длиной менее 64 бит (0 — короткие пакеты отбрасываются, 1 — адаптер обрабатывает короткие пакеты);
- бит 2 (AB) управляет обработкой пакетов с широковещательными адресами (0 — пакеты с широковещательным адресом отбрасываются, 1 — адаптер обрабатывает пакеты с широковещательным адресом);
- бит 3 (AM) управляет обработкой пакетов с групповыми адресами (0 — пакеты с групповым адресом не проверяются, 1 — адаптер проверяет пакеты с широковещательным адресом при помощи массива хеширования);
- бит 4 (PRO) управляет обработкой пакетов с физическими адресами (0 — принимаются только пакеты с физическим адресом, совпадающим с записанным в регистрах PAR0–PAR5 сетевого адаптера; 1 — принимаются пакеты с любыми физическими адресами);
- бит 5 (MON) позволяет переключить контроллер в режим «наблюдения» (0 — обычный режим, в котором принятые пакеты сохраняются в буферной памяти; 1 — режим наблюдения, в котором пакеты проверяются на наличие ошибок адресации, CRC и выравнивания кадров, но в памяти не сохраняются);
- биты 6 и 7 не используются (зарезервированы).

ПРИМЕЧАНИЕ

Биты AB и AM должны устанавливаться и сбрасываться совместно друг с другом. Установка в 1 всех трех битов управления контролем адресов (AB, AM и PRO) и всех разрядов в регистрах многоцелевого адреса (MAR0–MAR7) приводит к переключению адаптера в режим «подслушивания сети», когда принимаются любые обнаруженные в линии связи пакеты — вне зависимости от того, какой станции сети они адресованы.

Регистр состояния приемника (Receive Status Register, сокращенно **RSR**) отражает текущее состояние приемника.



7	6	5	4	3	2	1	0
DFR	DIS	PHY	MPA	FO	FAE	CRC	PRX

Рис. 9.8. Структура регистра состояния приемника RSR

Структура регистра RSR показана на рис. 9.8. Разряды регистра имеют следующее назначение:

- бит 0 (PRX) — признак успешного завершения приема пакета (устанавливается в 1, если пакет принят без ошибок);
- бит 1 (CRC) — признак ошибки CRC (устанавливается в 1, если при приеме пакета обнаружено несовпадение контрольного кода или ошибка выравнивания кадра; установка этого бита приводит к увеличению на 1 значения счетчика ошибок CNTR1);
- бит 2 (FAE) — признак ошибки выравнивания кадра (устанавливается в 1, если длина принятого пакета не выровнена на целое число байтов; установка этого бита приводит к увеличению на 1 значения счетчика ошибок CNTR0);
- бит 3 (FO) — признак переполнения FIFO (устанавливается в 1, если при приеме пакета произошло переполнение буфера FIFO; операция приема пакета в этом случае завершается аварийно);
- бит 4 (MPA) — признак потери пакета (устанавливается в 1 в том случае, если адаптер находится в режиме наблюдения и не сохраняет в памяти поступивший пакет, либо в том случае, когда пакет не был принят в результате переполнения буферной памяти; значение счетчика ошибок CNTR2 при этом увеличивается на 1);
- бит 5 (PHY) — индикатор типа адреса принятого пакета (0 — физический адрес, 1 — групповой или широковещательный адрес);
- бит 6 (DIS) — признак блокировки приемника (устанавливается в 1, когда работа приемника приостанавливается в результате переключения адаптера в режим наблюдения);

- бит 7 (DFR) — признак удлиненного пакета (устанавливается в 1, если длина пакета превышает 1518 байт).

Биты признака ошибки CRC и FAE используются совместно. Расшифровка значения кода, записываемого в эти биты, приведена в табл. 9.7.

Таблица 9.7. Значение кода ошибки, записываемого в биты CRC и FAE регистра состояния приемника RSR

Код ошибки		Тип ошибки
FAE	**CRC**	
0	0	Нет ошибок
0	1	Ошибка CRC
1	0	Запрещенная комбинация (не используется)
1	1	Ошибка CRC и ошибка выравнивания кадра

Передача данных внутри NE2000-совместимого адаптера осуществляется при помощи встроенного адаптера прямого доступа к памяти, который имеет собственный набор управляющих регистров (**регистров DMA**). Группу регистров DMA принято разделять на три подгруппы: подгруппу регистров передатчика (Transmit DMA Registers), подгруппу регистров приемника (Receive DMA Registers) и подгруппу регистров внешних операций (Remote DMA Registers).

Регистры DMA являются 16-разрядными, а для доступа к ним используется 8-разрядный режим, поэтому каждый регистр разделен на две половины: старший и младший байты.

В подгруппу регистров передатчика входят Регистр начальной страницы области передачи и Регистр счетчика передаваемых байтов.

Регистр начальной страницы области передачи (Transmit Page Start Register, сокращенно **TPSR**) содержит старшие 8 разрядов начального адреса области передачи (биты A8–A15). Встроенная оперативная память адаптера разделена на страницы по 256 байт, а начало пакета всегда выравнивается на начало страницы, поэтому младшие 8 разрядов начального адреса области передачи (биты A0–A7) сброшены в 0.

Регистр счетчика передаваемых байтов (Transmit Byte Count Register) разделен на младший (**TBCR0**) и старший (**TBCR1**) байты. В этот счетчик загружается размер передаваемого пакета в байтах (включая поле адреса источника, поле адреса получателя, поле длины пакета данных и поле данных). Загруженное в счетчик значение не должно превышать 1500 байт.

В подгруппу регистров приемника входят Регистр начальной страницы области приема, Регистр конечной страницы области приема, Регистр контроля границы, Регистр текущей страницы и Регистр текущего локального адреса DMA.

Регистр начальной страницы области приема (Page Start Register, сокращенно **PSTART**) содержит старшие 8 разрядов начального адреса кольцевого буфера приемника (биты A8–A15). Начало пакета всегда выравнивается на начало страницы, поэтому младшие 8 разрядов начального адреса кольцевого буфера (биты A0–A7) сброшены в 0.

Регистр конечной страницы области приема (Page Stop Register, сокращенно **PSTOP**), как явствует из его названия, содержит номер конечной страницы кольцевого буфера приемника.

Регистр контроля границы (Boundary Register, сокращенно **BNRY**) содержит номер первой занятой страницы кольцевого буфера и служит для контроля переполнения кольцевого буфера памяти приемника (если при приеме пакета данных происходит пересечение границы, операция приема завершается аварийно).

Регистр текущей страницы (Current Page Register, сокращенно **CURR**) содержит номер первой свободной страницы кольцевого буфера, которая может быть использована для приема данных. Во время выполнения инициализации адаптера в регистр CURR должно быть записано то же самое значение, что и в регистр PSTART, то есть регистр текущей страницы должен указывать на начало кольцевого буфера. После завершения процесса инициализации содержимое регистра CURR используется логическими схемами управления кольцевым буфером и может быть изменено только самим адаптером: перезаписывать этот регистр нельзя до следующего сброса адаптера.

Регистр текущего локального адреса DMA (Current Local DMA Register) разделен на младший (**CLDA0**) и старший (**CLDA1**) байты. Этот регистр позволяет проконтролировать текущий адрес DMA (адрес ячейки памяти, к которой в данный момент осуществляется обращение).

В подгруппу регистров внешних операций входят Регистр начального адреса для операций внешнего доступа, Регистр счетчика байтов для операций внешнего доступа и Регистр текущего адреса DMA для операций внешнего доступа, которые используются при выполнении обмена данными между центральным процессором компьютера и встроенной памятью адаптера.

Регистр начального адреса для операций внешнего доступа (Remote Start Address Register) разделен на младший (**RSAR0**) и старший (**RSAR1**) байты. Этот регистр задает начальный адрес для передачи блока данных.

Регистр счетчика байтов для операций внешнего доступа (Remote Byte Count Register) разделен на младший (**RBCR0**) и старший (**RBCR1**) байты. Этот регистр задает размер передаваемого блока данных в байтах.

Регистр текущего адреса DMA для операций внешнего доступа (Current Remote DMA Address) также разделен на младший (**CRDA0**) и старший (**CRDA1**) байты. Он доступен только для считывания и предназначен для контроля процесса передачи данных.

В группу **регистров физического адреса адаптера** (Physical Address Registers) входят шесть 8-разрядных регистров **PAR0–PAR5**. В эти регистры должен быть записан физический 48-разрядный адрес адаптера в сети Ethernet (регистр PAR0 при этом соответствует младшему байту адреса, регистр PAR5 — старшему байту адреса). Указанный адрес используется адаптером при принятии решения о приеме или игнорировании пакета.

В группу **регистров группового адреса адаптера** (Multicast Address Registers) входят восемь 8-разрядных регистров **MAR0–MAR7**. Регистры группового адреса предназначены для фильтрации адресов по контрольной сумме — в момент приема последнего бита адреса получателя пакета. Текущее значение счетчика контрольной суммы фиксируется в регистре-защелке, и старшие 6 разрядов полученного кода используются в качестве индекса элемента 64-разрядной битовой маски, записанной в регистрах MAR0–MAR7. Если использование групповых адресов разрешено и соответствующий значению индекса бит маски установлен в 1, пакет принимается, иначе — игнорируется.

В подгруппу счетчиков ошибок входят Счетчик ошибок выравнивания, Счетчик ошибок CRC и Счетчик потерянных пакетов. Восьмиразрядные регистры-счетчики имеют следующие общие свойства:

- максимальное значение количества ошибок каждого типа составляет 192;
- после считывания содержимого регистр-счетчик автоматически обнуляется.

Счетчик ошибок выравнивания (Frame Alignment Error Tally, сокращенно **CNTR0**) подсчитывает ошибки выравнивания кадра: значение счетчика увеличивается на единицу каждый раз, когда при приеме пакета возникает ошибка выравнивания.

Счетчик ошибок CRC (CRC Error Tally, сокращенно **CNTR1**) подсчитывает ошибки по контрольной сумме: значение счетчика увеличивается на единицу каждый раз, когда при приеме пакета обнаруживается несовпадение контрольной суммы.

Счетчик потерянных пакетов (Frame Lost Tally Register, сокращенно **CNTR2**) подсчитывает ошибки переполнения буфера памяти: значение счетчика увеличивается на единицу каждый раз, когда по причине отсутствия места в кольцевом буфере происходит потеря принимаемого пакета. Если адаптер работает в режиме монитора (отслеживает работу сети), регистр CNTR2 подсчитывает количество распознанных пакетов.

Регистр очереди данных (FIFO) позволяет контролировать содержимое очереди передаваемых данных при выполнении процедуры самодиагностики сетевого адаптера. Очередь данных имеет размер 8 байт, и для извлечения всей информации из очереди нужно повторить операцию считывания из регистра FIFO восемь раз.

ПРИМЕЧАНИЕ

Наличие регистра FIFO не является обязательным: у некоторых моделей NE2000-совместимых адаптеров регистр FIFO отсутствует.

Счетчик коллизий (Number of Collisions, сокращенно **NCR**) используется для подсчета количества коллизий, имевших место в процессе передачи пакета: значение счетчика увеличивается на единицу при обнаружении очередной коллизии. Структура регистра счетчика коллизий показана на рис. 9.9: используются только 4 младших разряда, поэтому значение счетчика не может превышать 15.

7	6	5	4	3	2	1	0
—	—	—	—	NC3	NC2	NC1	NC0

Рис. 9.9. Структура регистра счетчика коллизий NCR

Регистр ввода-вывода данных (IO Register) используется центральным процессором компьютера для доступа к оперативной памяти и ПЗУ адаптера. Регистр имеет смещение 10h от начала пространства ввода-вывода. Разрядность регистра составляет 8 или 16 бит — в зависимости от ширины шины данных, заданной в регистр DCR.

Чтобы получить доступ к определенному участку памяти адаптера, нужно загрузить начальный адрес этого участка в Регистр начального адреса для операций внешнего доступа RSAR, а количество передаваемых байтов — в Регистр счетчика байтов для операций внешнего

доступа RBCR. После каждой операции ввода-вывода текущий адрес будет увеличиваться на 1 (если заданная ширина шины данных составляет 8 разрядов) или на 2 (если ширина шины — 16 разрядов). Значение счетчика байтов будет уменьшаться соответственно на 1 или на 2 (при использовании 16-разрядной шины всегда должно передаваться четное число байтов).

Определение параметров сетевого адаптера

Для того чтобы определить параметры конфигурации сетевого адаптера, нужно выполнить операцию поиска адаптера на шине PCI по коду класса при помощи функции B103h PCI BIOS: код базового класса для адаптеров сети Ethernet имеет значение 02h, код подкласса — 00h, код интерфейса — 00h.

Используя полученный в результате поиска адрес устройства на шине PCI, программист может определить следующие параметры NE2000-совместимого адаптера:

- идентификатор изготовителя;
- идентификатор устройства;
- базовый адрес пространства ввода-вывода;
- номер используемой адаптером линии прерывания.

Идентификатор изготовителя и идентификатор устройства позволяют определить тип микросхемы, на основе которой построен адаптер. Таким способом можно удостовериться в том, что адаптер является NE2000-совместимым, и определить дополнительные (нестандартные) возможности адаптера. В Интернете можно найти документацию по используемым в настоящее время микросхемам RTL8029AS фирмы REALTEK (код изготовителя 10ECh, код устройства 8029h) [88] и VT86C926 фирмы VIA Technologies (код изготовителя 1106h, код устройства 0926h) [97].

Базовый адрес пространства ввода-вывода позволяет получить доступ к регистровым страницам NE2000-совместимого адаптера. Извлечь значение адреса пространства ввода-вывода можно из Нулевого регистра базового адреса (32-разрядного регистра со смещением 10h от начала конфигурационного пространства устройства PCI).

Номер используемой адаптером линии IRQ можно извлечь из Регистра номера прерывания (8-разрядного регистра со смещением 3Ch от начала конфигурационного пространства).

Последовательность инициализации адаптера

Процедура инициализации адаптера включает представленную ниже последовательность операций [99].

- Настроить Регистр команды CR на страницу 0, записав в него значение 21h.
- Настроить Регистр управления форматом данных DCR (например, значение 49h соответствует режиму работы, в котором передача данных выполняется 16-разрядными словами, используется порядок передачи байтов, принятый в процессорах 80x86, адаптер находится в рабочем режиме, автоинициализация не используется, а размер очереди данных составляет 8 байт).
- Обнулить Регистр начального адреса для операций внешнего доступа, записав значение 0 в регистры RBCR0 и RBCR1.
- Настроить Регистр управления приемником RCR, записав в него значение 1Ch, соответствующее обычному режиму работы (пакеты с ошибками не сохраняются, пакеты недопустимой длины отбрасываются, обрабатываются адреса любого типа).
- Перевести адаптер в режим самодиагностики 1 или 2, поместив в Регистр команды значение 02h или 04h.
- Настроить группу регистров приемника, задав границы кольцевого буфера при помощи регистров BNDRY, PSTART и PSTOP.
- Обнулить Регистр статуса прерывания ISR, записав в него значение FFh.
- Настроить Регистр маскирования прерываний IMR.
- Настроить Регистр команды CR на страницу 1, записав в него значение 61h.
- Настроить Регистры физического адреса.
- Настроить Регистр текущей страницы CURR, записав в него то же самое значение, что и в регистр PSTART.
- Настроить переключиться на страницу 0 и активизировать адаптер, записав в Регистр команды значение 22h.

Внутреннее адресное пространство адаптера

Организация внутреннего адресного пространства NE2000-совместимого сетевого адаптера зависит от настройки регистра управления

форматом данных DCR. Ниже мы будем рассматривать только вариант организации, соответствующий 16-разрядному режиму передачи данных с использованием порядка байтов, принятого для процессоров 80x86.

Структура адресного пространства при использовании 16-разрядного режима передачи данных показана на рис. 9.10. Адресное пространство в этом случае распределяется следующим образом:

- по адресам 00h-1Fh отображается участок ПЗУ, в котором записан физический адрес сетевого адаптера, задаваемый фирмой-изготовителем;
- в диапазоне 4000h-7FFFh находится основная страница оперативной памяти контроллера размером 16 Кбайт.



Рис. 9.10. Структура внутреннего адресного пространства сетевого адаптера

Структура отображаемого участка ПЗУ также зависит от настройки регистра управления форматом данных DCR. В случае использования 16-разрядного режима передачи данных физический адрес адаптера отображается следующим образом:

- байт 0 физического адреса адаптера имеет смещение 0000h;
- байт 1 физического адреса адаптера имеет смещение 0002h;
- байт 2 физического адреса адаптера имеет смещение 0004h;
- байт 3 физического адреса адаптера имеет смещение 0006h;
- байт 4 физического адреса адаптера имеет смещение 0008h;
- байт 5 физического адреса адаптера имеет смещение 000Ah.

Для проверки правильности отображения данных ПЗУ в адресном пространстве используются специальные контрольные значения: по адресу **001Ch** должен находиться байт со значением 57h, по адресу 001Eh — также байт со значением 57h.

Прием и передача пакетов

Область оперативной памяти адаптера в процессе настройки должна быть поделена на две части, как показано на рис. 9.10: буфер данных передатчика и кольцевой буфер данных приемника.

С целью упрощения схемы управления адаптером встроенная оперативная память была разделена на страницы размером 256 байт. Принимаемый или передаваемый пакет всегда занимает целое количество таких страниц, причем начало пакета должно быть выровнено на начало страницы. Если в конце последней занимаемой пакетом страницы есть неиспользуемый участок, при выполнении операций с памятью он просто игнорируется (не обрабатывается).

Буфер данных передатчика размещается в начальном участке области оперативной памяти и обычно занимает область размером 1,5 Кбайт (6 страниц по 256 байт), соответствующую максимальному размеру пакета данных.

Адаптер работает с кадрами Ethernet на уровне управления доступом к сети (Media Access Control, сокращенно MAC), поэтому структура пакета данных, загружаемого в память передатчика, соответствует структуре кадров Novell 802.3 и Ethernet II [19]. Формат пакета данных описан в табл. 9.8.

Правила работы сети Ethernet не допускают передачи пакетов размером менее 64 байт, поэтому если размер области данных меньше 46 байт, в пакет вставляется так называемое «набивочное» поле, содержащее байты-заполнители (обычно — нули). Если размер области данных превышает 46 байт, набивочное поле в пакет не вставляется.

Таблица 9.8. Формат пакета данных NE2000-совместимого сетевого адаптера

Смещение	Размер	Назначение поля
0	6 байт	Адрес получателя (Destination Address)
6	6 байт	Адрес отправителя (Source Address)
12	WORD	Тип пакета или размер области данных (Type/Length)

продолжение ⇒

Таблица 9.8 (*продолжение*)

Смещение	Размер	Назначение поля
14	n байт	Область данных (Data)
14 + n	46 – n байт	Набивочное поле (Pad)

Для записи пакета в буфер передатчика нужно загрузить номер начальной страницы буфера передатчика в Регистр начального адреса для операций внешнего доступа RSAR, а размер пакета в байтах — в Регистр счетчика байтов для операций внешнего доступа RBCR. Далее нужно произвести загрузку пакета через регистр ввода-вывода по байтам или по словам.

После того, как передаваемый пакет записан в буфер, нужно указать передатчику номер начальной страницы буфера, загрузив его в Регистр начальной страницы области передачи TPSR; размер передаваемого пакета нужно занести в Регистр счетчика передаваемых байтов TBCR. Для запуска процесса передачи пакета нужно занести значение 26h в регистр команды CR.

Прием пакетов осуществляется адаптером по мере их поступления. Принятые пакеты заносятся в буфер приемника.

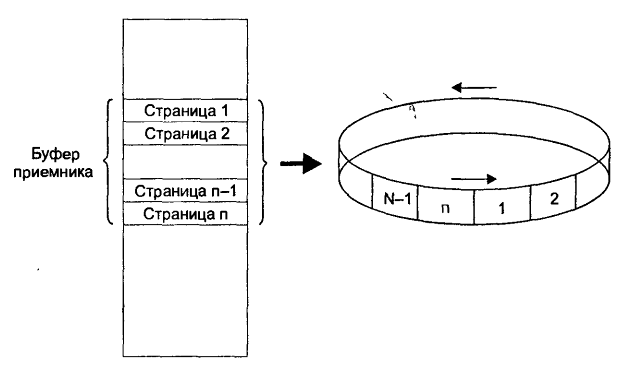


Рис. 9.11. Структура кольцевого буфера данных приемника

Буфер данных приемника размещается вслед за буфером передатчика и организован в виде кольца, как показано на рис. 9.11. После инициализации адаптера регистры PSTART, BNDRY и CURR указывают на первую страницу кольцевого буфера, а регистр PSTOP

указывает на страницу n + 1, которая находится за пределами буфера и за пределами оперативной памяти.

Каждый принятый пакет может занимать одну или несколько страниц по 256 байт. Если в процессе приема достигнута страница кольцевого буфера с номером n, то следующей будет использоваться страница 1.

После завершения приема пакета в регистр CURR будет записан номер первой свободной страницы. Если в процессе приема пакета происходит пересечение страницы, на которую указывает регистр BNDRY, фиксируется ошибка переполнения буфера.

Регистр BNRDY указывает на страницу, содержащую начало первого непрочитанного (не переписанного из кольцевого буфера в оперативную память компьютера) пакета. Когда центральный процессор завершает операцию считывания очередного пакета из кольцевого буфера, он должен записать в регистр BNDRY значение, соответствующее началу следующего пакета, после чего занимаемая считанным пакетом область памяти считается освобожденной и доступна для записи других пакетов.

Перед началом записи пакета в кольцевой буфер контроллер сетевого адаптера заносит в буфер 4-байтную структуру-описатель пакета. Она включает в себя:

- байт статуса (Receive Status), в который переписывается информация из регистра состояния приемника;
- байт номера начальной страницы следующего пакета (Next Packet Pointer);
- 16-разрядное слово, содержащее общий размер пакета вместе со структурой-описателем (Receive Byte Count).

Таким образом, в буфере приемника пакет данных размещается не с начала страницы, а смещен на 4 байта, как показано на рис. 9.12.

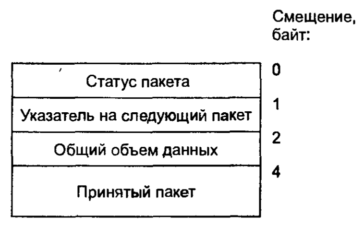


Рис. 9.12. Размещение принятого пакета в буфере приемника

Перед началом считывания пакета данных нужно извлечь номер начальной страницы пакета из Регистра указателя границы BNDRY, запомнить его и занести в старший байт Регистра начального адреса для операций внешнего доступа RSAR (младший байт RSAR нужно обнулить).

Прежде чем начать выполнение операции считывания пакета из кольцевого буфера, программист должен извлечь и проанализировать структуру-описатель пакета. Для считывания структуры-описателя пакета нужно загрузить значение 4 в регистр RBCR, после чего считать через Регистр ввода-вывода 4 байта или 2 слова (в зависимости от заданной разрядности шины данных).

Структура-описатель пакета обрабатывается следующим образом:

- байт статуса позволяет определить, насколько успешно выполнена операция приема пакета и были ли обнаружены какие-либо ошибки в процессе сбоя. Обрабатывать байт статуса программисту требуется только в том случае, если разрешен прием пакетов с ошибками;
- байт номера начальной страницы следующего пакета следует запомнить и после завершения считывания пакета переписать его значение в регистр границы BNDRY;
- из общего размера пакета нужно вычесть размер описателя (4 байта), после чего запомнить размер пакета в оперативной памяти компьютера и загрузить его в Регистр счетчика байтов для операций внешнего доступа RBCR (перед началом выполнения операции считывания пакета).

Далее выполняется операция считывания полученного пакета через Регистр ввода-вывода. Количество операций считывания при использовании побайтной передачи равно значению размера пакета, а при передаче 16-разрядными словами — размеру пакета, деленному на 2.

В кольцевом буфере, вообще говоря, может находиться одновременно несколько пакетов данных. Признаком того, что из буфера прочитаны все пакеты и можно прекратить процесс считывания, является совпадение значений в регистрах CURR и BNDRY.

Листинг 9.1 содержит набор процедур общего назначения, предназначенных для работы с NE2000-совместимыми сетевыми адаптерами:

- процедура `SearchEthernetContr` предназначена для поиска адаптера Ethernet на шине PCI при помощи функций PCI BIOS;
- процедура `InitializeAdapter` выполняет инициализацию регистров адаптера;

- процедура PCtoNIC предназначена для копирования блока данных из оперативной памяти компьютера во встроенную память сетевого адаптера;
- процедура NICtoPC предназначена для копирования блока данных из памяти сетевого адаптера компьютера в оперативную память компьютера;
- процедура SendPacket использует процедуру PCtoNIC для загрузки пакета в буфер передатчика, а затем запускает процесс передачи;
- процедура GetPacket использует процедуру NICtoPC для считывания принятого пакета из кольцевого буфера;
- процедура SetEthernertAddress копирует физический адрес из ПЗУ в регистры физического адреса адаптера;

Листинг 9.1. Набор процедур для работы с NE2000-совместимыми адаптерами Ethernet по шине PCI

```
; СМЕЩЕНИЕ РЕГИСТРОВ АДАПТЕРА ОТ БАЗОВОГО АДРЕСА
; Регистры страницы 0
PAGESTART            equ 01h ;запись
PAGESTOP             equ 02h ;запись
BOUNDARY             equ 03h ;запись
TRANSMITSTATUS       equ 04h ;чтение
TRANSMITPAGE         equ 04h ;запись
TRANSMITBYTECOUNT0   equ 05h ;запись
INTERRUPTSTATUS      equ 07h ;чтение и запись
REMOTESTARTADDRESS0  equ 08h ;запись
REMOTEBYTECOUNT0     equ 0Ah ;запись
RECEIVESTATUS        equ 0Ch ;чтение
RECEIVECONFIGURATION equ 0Ch ;запись
TRANSMITCONFIGURATION equ 0Dh ;запись
DATACONFIGURATION    equ 0Eh ;запись
INTERRUPTMASK        equ 0Fh ;запись
; Регистры страницы 1
CURRENT              equ 07h ;чтение и запись
; СМЕЩЕНИЕ ПОРТА ВВОДА-ВЫВОДА
IOPORT               equ 10h ;чтение и запись

; ПАРАМЕТРЫ ДЛЯ НАСТРОЙКИ РЕГИСТРОВ УПРАВЛЕНИЯ
; Параметры настройки регистра управления форматом
; данных
V_DCR equ 49h
; Параметры настройки регистра управления приемником
V_RCR equ 1Ch
; Параметры настройки регистра управления передатчиком
V_TCR equ 0
; Параметры настройки регистра маскирования прерываний
```

продолжение ⇨

Листинг 9.1 (*продолжение*)

```
; (прерывания запрещены)
V_IMR equ 0

; СТРАНИЧНЫЕ ГРАНИЦЫ БУФЕРОВ В ПАМЯТИ АДАПТЕРА
; Адрес буфера передачи (буфер занимает 6 страниц
; по 256 байт и его общий объем составляет 1536 байт)
TRANSMITBUFFER equ 40h
; Адрес начальной страницы кольцевого буфера приема
PSTART         equ 46h
; Верхняя граница кольцевого буфера приема
PSTOP          equ 80h

DATASEG
; СООБЩЕНИЯ ОБ ОШИБКАХ
NoPCI  DB 12,24,"Система не поддерживает PCI BIOS",0
NoEth  DB 12,10,"NE2000-совместимый адаптер Ethernet "
       DB "на шине PCI не обнаружен",0
BadRg  DB 12,28,"Неверный номер регистра",0
NoPack DB 12,25,"В кольцевом буфере нет пакета",0
; ПАРАМЕТРЫ АДАПТЕРА, СВЯЗАННЫЕ С ШИНОЙ PCI
; Координаты сетевой карты на шине PCI
EthBusNumber    DB ? ;номер шины
EthDeviceNumber DB ? ;номер устройства и номер функции
; Идентификаторы устройства PCI
EthVendorID     DW ? ;идентификатор изготовителя
EthDeviceID     DW ? ;идентификатор устройства
; Адрес блока регистров адаптера Ethernet
EthBaseAddr     DW ?
; Номер используемого прерывания IRQ
EthIntLine      DB ?
; ПЕРЕМЕННЫЕ И МАССИВЫ
; Начальная страница принимаемого пакета
PackStartPage   DB ?
; Размер принятого пакета в байтах
RecPackSize     DW ?
; Количество подлежащих считыванию байтов
RemBytes        DW ?
; Буфер для хранения физического адреса
PhisicalAddress DB 32 DUP(?)
; Буфер для формирования передаваемого пакета
DataOutBuffer   DB 6*256 DUP(?)
; Буфер для сохранения принятого пакета
DataInBuffer    DB 6*256 DUP(?)
ENDS

CODESEG
;*************************************************
;*   НАЙТИ НА ШИНЕ PCI СЕТЕВОЙ АДАПТЕР ETHERNET   *
;*           И ОПРЕДЕЛИТЬ ЕГО ПАРАМЕТРЫ           *
```

```
;* Процедура не имеет входных параметров.        *
;* В случае успешного завершения операции поиска  *
;* параметры адаптера сохраняются в глобальных    *
;* переменных EthBusNumber, EthDeviceNumber,      *
;* EthVendorID, EthDeviceID, EthBaseAddr и        *
;* EthIntLine.                                    *
;**************************************************
PROC SearchEthernetContr near
        pushad
; Проверить наличие PCI BIOS
        mov     AX,0B101h
        int     1Ah
        jc      @@PCIBIOSNotFound
        cmp     EDX,20494350h
        jne     @@PCIBIOSNotFound
; Найти адаптер Ethernet по коду класса
        mov     AX,0B103h
        mov     ECX,020000h
        mov     SI,0
        int     1Ah
        jnc     @@ReadPCIRegisters ;устройство найдено
; Адаптер Ethernet не найден
        MFatalError NoEth
; Устройство обнаружено, его координаты на шине PCI
; находятся в регистре BX
@@ReadPCIRegisters:
; Запомнить координаты адаптера
        mov     [EthBusNumber],BH
        mov     [EthDeviceNumber],BL
; Получить идентификатор изготовителя
        mov     AX,0B109h ;читать слово
        mov     DI,0      ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [EthVendorID],CX
; Получить идентификатор устройства
        mov     AX,0B109h ;читать слово
        mov     DI,2      ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [EthDeviceID],CX
; Получить базовый адрес блока регистров
; адаптера Ethernet
        mov     AX,0B10Ah ;читать двойное слово
        mov     DI,10h    ;смещение слова
        int     1Ah
        jc      @@BadRegisterNumber
        ; Обнулить 5 младших бит
        and     CX,0FFE0h
        ; Сохранить только младшее слово адреса
```

продолжение ⇨

Листинг 9.1 (*продолжение*)

```
        mov     [EthBaseAddr],CX
; Получить номер используемого устройством
; прерывания IRQ
        mov     AX,0B108h ;читать байт
        mov     DI,3Ch    ;смещение байта
        int     1Ah
        jc      @@BadRegisterNumber
        mov     [EthIntLine],CL
        popad
        ret

; Обработка ошибок
@@PCIBIOSNotFound:
        ; Не поддерживается PCI BIOS
        MFatalError NoPCI
@@BadRegisterNumber:
        ; Неверный номер регистра
        MFatalError BadRg
ENDP SearchEthernetContr

;*****************************************************
;*      ИНИЦИАЛИЗАЦИЯ NE2000-СОВМЕСТИМОГО АДАПТЕРА    *
;* Буфер передачи размещается в диапазоне 4000h-45FFh, *
;* буфер приема - в диапазоне 4600h-7FFFh.            *
;* Входные параметры:                                 *
;* EthBaseAddr - базовый адрес блока регистров.       *
;*****************************************************
PROC InitializeAdapter near
        push    AX
        push    DX
; Приостановить работу адаптера и установить нулевую
; регистровую страницу
        mov     DX,[EthBaseAddr]
        mov     AL,21h
        out     DX,AL
; Инициализировать регистр управления форматом данных
        mov     DX,[EthBaseAddr]
        add     DX,DATACONFIGURATION
        mov     AL,V_DCR
        out     DX,AL
; Обнулить регистр счетчика байтов для внешнего доступа
        mov     DX,[EthBaseAddr]
        add     DX,REMOTEBYTECOUNT0
        xor     AL,AL
        out     DX,AL   ;младший байт
        inc     DX
        out     DX,AL   ;старший байт
; Настроить регистр управления приемником
        mov     DX,[EthBaseAddr]
```

```
        add     DX,RECEIVECONFIGURATION
        mov     AL,V_RCR
        out     DX,AL
; Задать начальную страницу буфера передачи
        mov     DX,[EthBaseAddr]
        add     DX,TRANSMITPAGE ;transmit page start
        mov     AL,TRANSMITBUFFER
        out     DX,AL
; Временно перейти в режим самотестирования
        mov     DX,[EthBaseAddr]
        add     DX,TRANSMITCONFIGURATION
        mov     AL,02
        out     DX,AL
; Задать начальную страницу и текущую границу
; кольцевого буфера
        mov     DX,[EthBaseAddr]
        add     DX,PAGESTART
        mov     AL,PSTART
        out     DX,AL
        add     DX,2
        out     DX,AL
; Задать конечную страницу кольцевого буфера
        mov     DX,[EthBaseAddr]
        add     DX,PAGESTOP
        mov     AL,PSTOP
        out     DX,AL
; Перейти на регистровую страницу 1
        mov     DX,[EthBaseAddr]
        mov     AL,61h
        out     DX,AL
; Задать номер текущей страницы кольцевого буфера
        mov     DX,[EthBaseAddr]
        add     DX,CURRENT
        mov     AL,PSTART
        out     DX,AL
; Перейти на регистровую страницу 0 и активизировать
; адаптер
        mov     DX,[EthBaseAddr]
        mov     AL,22h
        out     DX,AL
; Сбросить все признаки прерываний
        mov     DX,[EthBaseAddr]
        add     DX,INTERRUPTSTATUS
        mov     AL,0FFh
        out     DX,AL
; Настроить регистр маскирования прерываний
        mov     DX,[EthBaseAddr]
        add     DX,INTERRUPTMASK
        mov     AL,V_IMR
        out     DX,AL
; Настроить регистр управления передатчиком
```

продолжение ⇗

Листинг 9.1 (*продолжение*)

```
        mov     DX,[EthBaseAddr]
        add     DX,TRANSMITCONFIGURATION
        mov     AL,V_TCR
        out     DX,AL
        pop     DX
        pop     AX
        ret
ENDP InitializeAdapter

;*********************************************
;*            ПЕРЕДАТЬ ПАКЕТ ДАННЫХ          *
;* Входные параметры:                        *
;* CX - общее количество байтов в пакете (включая *
;*      заголовок);                          *
;* DS:SI - указатель на область памяти, в которой *
;*         размещен передаваемый пакет;      *
;* EthBaseAddr - базовый адрес блока регистров.  *
;*********************************************
PROC SendPacket near
        pusha
; Цикл ожидания готовности передатчика
@@Wait: mov     DX,[EthBaseAddr]
        in      AL,DX
        cmp     AL,26h ;передатчик занят?
        je      @@Wait
        cli             ;запретить прерывания
        push    CX      ;сохранить количество байтов
; Загрузить пакет данных в память адаптера
        mov     AH,TRANSMITBUFFER
        xor     AL,AL
        call    PCtoNIC
; Задать передатчику номер начальной страницы
        mov     DX,[EthBaseAddr]
        add     DX,TRANSMITPAGE
        mov     AL,TRANSMITBUFFER
        out     DX,AL
        pop     CX
; Загрузить регистр счетчика передаваемых байтов
        mov     DX,[EthBaseAddr]
        add     DX,TRANSMITBYTECOUNT0
        mov     AL,CL  ;младший байт
        out     DX,AL
        inc     DX
        mov     AL,CH  ;старший байт
        out     DX,AL
; Начать передачу пакета
        mov     DX,[EthBaseAddr]
        mov     AL,26h
        out     DX,AL
```

```
        sti             ;разрешить прерывания
        popa
        ret
ENDP SendPacket

;************************************************
;*    ПЕРЕПИСАТЬ ПЕРЕДАВАЕМЫЙ ПАКЕТ ИЗ ПАМЯТИ     *
;*     КОМПЬЮТЕРА В ПАМЯТЬ СЕТЕВОГО АДАПТЕРА     *
;* Входные параметры:                            *
;* AX - номер страницы буфера адаптера, с которой *
;*      начинается запись пакета;                *
;* CX - общее количество байтов в пакете (включая *
;*      заголовок);                              *
;* DS:SI - указатель на область памяти, в которой *
;*         размещен передаваемый пакет;          *
;* EthBaseAddr - базовый адрес блока регистров.  *
;************************************************
PROC PCtoNIC near
        push    AX      ;сохранить номер страницы
; Округлить количество байтов пакета до четного числа
        inc     CX
        and     CX,0FFFEh
; Загрузить счетчик байтов
        mov     DX,[EthBaseAddr]
        add     DX,REMOTEBYTECOUNT0
        mov     AL,CL  ;младший байт
        out     DX,AL
        inc     DX
        mov     AL,CH  ;старший байт
        out     DX,AL
        pop     AX      ;восстановить номер страницы
; Загрузить номер страницы буфера
        mov     DX,[EthBaseAddr]
        add     DX,REMOTESTARTADDRESS0
        out     DX,AL  ;младший байт
        inc     DX
        mov     AL,AH
        out     DX,AL  ;старший байт
; Подать команду загрузки данных в память адаптера
        mov     DX,[EthBaseAddr]
        mov     AL,12h
        out     DX,AL
; Цикл записи данных по словам
        mov     DX,[EthBaseAddr]
        add     DX,IOPORT ;порт ввода-вывода
        shr     CX,1      ;разделить на 2
@@WritingWord:
        lodsw
        out     DX,AX
        loop    @@WritingWord
```

продолжение ⇨

Листинг 9.1 (*продолжение*)

```
; Настройть DX на регистр статуса прерывания
        mov     DX,[EthBaseAddr]
        add     DX,INTERRUPTSTATUS
; Ожидать завершения внутренней передачи данных
        mov     CX,0
@@CheckDMA:
        in      AL,DX
        test    AL,40h    ;цикл DMA завершен?
        jnz     @@End
        loop    @@CheckDMA
@@End:
; Сбросить признак прерывания DMA в регистре статуса
; прерывания
        mov     AL,40h
        out     DX,AL
        clc
        ret
ENDP PCtoNIC

;***************************************************
;*   ПЕРЕДАТЬ ПРИНЯТЫЙ ПАКЕТ ИЗ ПАМЯТИ АДАПТЕРА    *
;*        В ОПЕРАТИВНУЮ ПАМЯТЬ КОМПЬЮТЕРА          *
;* Входные параметры:                              *
;* AX - номер страницы буфера адаптера, с которой *
;*      начинается считывание пакета;              *
;* CX - общее количество байтов в пакете (включая *
;*      заголовок);                                *
;* ES:DI - указатель на область памяти, в которую *
;*         должен быть записан принятый пакет;     *
;* EthBaseAddr - базовый адрес блока регистров.    *
;***************************************************
PROC NICtoPC near
        push    AX      ;сохранить номер страницы
; Сделать значение счетчика четным
        inc     CX
        and     CX,0FFFEh
; Загрузить счетчик байтов
        mov     DX,[EthBaseAddr]
        add     DX,REMOTEBYTECOUNT0
        mov     AL,CL
        out     DX,AL  ;младший байт
        inc     DX
        mov     AL,CH
        out     DX,AL  ;старший байт
        pop     AX      ;восстановить номер страницы
; Загрузить номер страницы буфера
        mov     DX,[EthBaseAddr]
        add     DX,REMOTESTARTADDRESS0
```

```
        out     DX,AL  ;младший байт
        inc     DX
        mov     AL,AH
        out     DX,AL  ;старший байт
; Подать команду считывания данных из памяти адаптера
        mov     DX,[EthBaseAddr]
        mov     AL,0Ah
        out     DX,AL
; Цикл чтения данных по словам
        mov     DX,[EthBaseAddr]
        add     DX,IOPORT ;порт ввода-вывода
        shr     CX,1      ;разделить на 2
@@ReadingWord:
        in      AX,DX
        stosw
        loop    @@ReadingWord
; Настроить DX на регистр статуса прерывания
        mov     DX,[EthBaseAddr]
        add     DX,INTERRUPTSTATUS
; Ожидать завершения внутренней передачи данных
        mov     CX,0
@@CheckDMA:
        in      AL,DX
        test    AL,40h    ;цикл DMA завершен?
        jnz     @@End
        loop    @@CheckDMA
; Сбросить признак прерывания DMA в регистре статуса
; прерывания
@@End:  mov     AL,40h
        out     DX,AL
        ret
ENDP NICtoPC

;*************************************************
;*      ПРИНЯТЬ ПАКЕТ ИЗ КОЛЬЦЕВОГО БУФЕРА       *
;* Входные параметры:                            *
;* EthBaseAddr - базовый адрес блока регистров.  *
;* Выходные параметры:                           *
;*************************************************
PROC GetPacket near
        pusha
        push    ES
; Проверить наличие сигнала приема пакета
        mov     DX,[EthBaseAddr]
        add     DX,INTERRUPTSTATUS
        in      AL,DX
        test    AL,01h        ;получен пакет?
        jz      @@Err
; Сбросить сигнал приема пакета
        mov     AL,01h
```

продолжение ⇗

Листинг 9.1 (*продолжение*)

```
        out     DX,AL
; Создать указатель на буфер для приема пакета
        mov     AX,DS
        mov     ES,AX
        mov     DI,offset DataInBuffer
; Определить начальную страницу пакета
        mov     DX,[EthBaseAddr]
        add     DX,BOUNDARY
        in      AL,DX
        mov     [PackStartPage],AL
        mov     AH,AL
        xor     AL,AL
; Принять первый 256-байтный блок данных
        mov     CX,256
        call    NICtoPC
; Вычислить полный размер пакета
        mov     CX,[word ptr DataInBuffer+2]
        mov     [RecPackSize],CX       ;запомнить размер
        mov     [RemBytes],CX
; Цикл чтения 256-байтных страниц
@@GetPack:
        ; Вычислить следующую страницу
        inc     [PackStartPage]
        cmp     [PackStartPage],PSTOP ;конец буфера?
        jb      @@ReadNextPage
        mov     [PackStartPage],PSTART
@@ReadNextPage:
        cmp     [RemBytes],256
        jbe     @@EndOfPack
        sub     [RemBytes],256
; Принять очередной блок данных
        mov     AH,[PackStartPage]
        xor     AL,AL
        mov     CX,256
        call    NICtoPC
        jmp     @@GetPack

@@EndOfPack:
        mov     DX,[EthBaseAddr]
        add     DX,BOUNDARY
        mov     AL,[PackStartPage]
        out     DX,AL
        pop     ES
        popa
        ret
; Обработка ошибок
@@Err:  MFatalError NoPack
ENDP GetPacket
```

```
;*********************************************
;* УСТАНОВИТЬ ФИЗИЧЕСКИЙ АДРЕС И ГРУППОВОЙ АДРЕС *
;* Входные параметры:                          *
;* EthBaseAddr - базовый адрес блока регистров.  *
;*********************************************
PROC SetEthernertAddress near
        pusha
; Установить регистровую страницу 0
        mov     DX,[EthBaseAddr]
        mov     AL,21h
        out     DX,AL
; Получить данные о физическои адресе адаптера из ПЗУ
        push    ES
        mov     AX,DS
        mov     ES,AX
        mov     DI,offset PhisicalAddress
        mov     CX,32
        mov     AX,0
        call    NICtoPC
        pop     ES
; Установить регистровую страницу 1
        mov     DX,[EthBaseAddr]
        mov     AL,61h
        out     DX,AL
; Загрузить физический адрес
        mov     DX,[EthBaseAddr]
        inc     DX
        mov     AL,[PhisicalAddress]
        out     DX,AL
        inc     DX
        mov     AL,[PhisicalAddress+2]
        out     DX,AL
        inc     DX
        mov     AL,[PhisicalAddress+4]
        out     DX,AL
        inc     DX
        mov     AL,[PhisicalAddress+6]
        out     DX,AL
        inc     DX
        mov     AL,[PhisicalAddress+8]
        out     DX,AL
        inc     DX
        mov     AL,[PhisicalAddress+10]
        out     DX,AL
; Загрузить групповой адрес
        mov     AL,0FFh
        mov     DX,[EthBaseAddr]
        add     DX,08h
        out     DX,AL
        inc     DX
```

продолжение ⇨

Листинг 9.1 (*продолжение*)

```
        out     DX,AL
        inc     DX
        out     DX,AL
        inc     DX
        out     DX,AL
        inc     DX
        out     DX,AL
        inc     DX
        out     DX,AL
        inc     DX
        out     DX,AL
        inc     DX
        out     DX,AL
        popa
        ret
ENDP SetEthernertAddress
ENDS
```

Листинг 9.2 содержит программу TestEthernetContr, предназначенную для поиска NE2000-совместимого адаптера Ethernet по шине PCI. Программа использует процедуру поиска SearchEthernetContr из листинга 9.1 для поиска адаптера, а также универсальные процедуры ввода-вывода из листинга 1.2 для отображения параметров найденного устройства на экране монитора.

Листинг 9.2. Поиск NE2000-совместимого адаптера Ethernet по шине PCI и считывание его параметров

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"

DATASEG
; Текстовые сообщения
Txt0 DB LIGHTCYAN,0,18
     DB "ПОИСК АДАПТЕРА ETHERNET ПРИ ПОМОЩИ PCI BIOS",0
Txt1 DB 2,26,"Параметры адаптера Ethernet",0
     DB 4,28,"Номер шины:",0
     DB 5,22,"Номер устройства:",0
     DB 6,25,"Номер функции:",0
     DB 7,12,"Идентификатор изготовителя:",0
```

```
    DB 8,14,"Идентификатор устройства:",0
    DB 9,8,"Базовый адрес набора регистров:",0
    DB 10,8,"Номер используемого прерывания:",0
AnyK DB YELLOW,24,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;****************************
;* Основной модуль программы *
;****************************
PROC TestEthernetContr
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Вывести заголовок
        MShowColorString Txt0
; Установить зеленый цвет символов и черный фон
        mov     [TextColorAndBackground],LIGHTGREEN
; Найти адаптер Ethernet
        call    SearchEthernetContr
; Проверить тип адаптера
        cmp     [EthDeviceID],8029h ;RTL8029AS
        je      @@ShowParameters
        cmp     [EthDeviceID],0926h ;VT86C926
        jne     @@AdapterNotFound
@@ShowParameters:
; Вывести заголовки полей
        MShowText 8,Txt1
; Установить желтый цвет символов и черный фон
        mov     [TextColorAndBackground],YELLOW
; Вывести полученные данные на экран
        MShowHexByte 4,40,[EthBusNumber]
        mov     BL,[EthDeviceNumber]
        shr     BL,3
        MShowHexByte 5,40,BL
        mov     BL,[EthDeviceNumber]
        and     BL,111b
        MShowHexByte 6,40,BL
        MShowHexWord 7,40,[EthVendorID]
        MShowHexWord 8,40,[EthDeviceID]
```

продолжение ⇨

Листинг 9.2 (*продолжение*)

```
        MShowHexWord 9,40,[EthBaseAddr]
        MShowHexByte 10,40,[EthIntLine]
; Ожидать нажатия любой клавиши
        MShowColorString AnyK
        call    GetChar
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
; Обработка ошибок
@@BadRegisterNumber:
        MFatalError BadRg ;неверный номер регистра
@@AdapterNotFound:
        MFatalError NoEth ;адаптер Ethernet не найден
ENDP TestEthernetContr
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить процедуры для обслуживания работы
; контроллера Ethernet
include "list9_01.inc"

END
```

Любой сетевой адаптер может быть настроен на режим прослушивания сети. В режиме прослушивания осуществляется прием не только тех пакетов, которые предназначены данному конкретному адаптеру, но и всех остальных пакетов, передаваемых по сегменту сети, к которому подсоединен адаптер.

Листинг 9.3 содержит программу WaitAnyPacket, которая перехватывает один из передаваемых по сети пакетов (первый попавшийся) и отображает его содержимое на экране монитора в коде ASCII. Программа использует процедуры для работы с NE2000-совместимым адаптером из листинга 9.1, а также универсальные процедуры ввода-вывода из листинга 1.2 и процедуры перевода десятичных чисел из листинга 2.5.

Листинг 9.3. Программа, иллюстрирующая работу адаптера в режиме прослушивания сети

```
IDEAL
P386
LOCALS
```

```
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "listl_03.inc"
; Подключить файл макросов
include "listl_04.inc"

DATASEG
; Текстовые сообщения
Txt0 DB LIGHTCYAN,0,25,"ПРИЕМ ПАКЕТА ИЗ СЕТИ ETHERNET",0
Txt1 DB 2,0,"Размер пакета, байт:",0
     DB 3,0,"Адрес получателя:",0
     DB 4,0,"Адрес отправителя:",0
     DB 5,0,"Поле L/T:",0
     DB 6,0,"Поле данных:",0
     DB 24,35,"Ждите ...",0
AnyK DB YELLOW,24,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC WaitAnyPacket
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Найти контроллер Ethernet
        call    SearchEthernetContr
; Проверить тип адаптера
        cmp     [EthDeviceID],8029h ;RTL8029AS
        je      @@InitAdapter
        cmp     [EthDeviceID],0926h ;VTB6C926
        jne     @@AdapterNotFound
@@InitAdapter:
; Инициализировать сетевой контроллер
        call    InitializeAdapter
; Вывести текст
```

продолжение ⇨

Листинг 9.3 (*продолжение*)

```
        MShowColorString Txt0
        mov     [TextColorAndBackground],LIGHTGREEN
        MShowText 6,Txt1
        mov     [TextColorAndBackground],YELLOW
; Загрузить физический адрес и групповой адрес
        call    SetEthernertAddress
; Активизировать адаптер
        mov     AL,22h
        mov     DX,[EthBaseAddr]
        out     DX,AL
; Ожидать сигнал о поступлении пакета
@@WaitPacket:
        mov     DX,[EthBaseAddr]
        add     DX,INTERRUPTSTATUS
        in      AL,DX
        test    AL,01h   ;получен пакет?
        jz      @@WaitPacket
; Принять пакет
        call    GetPacket
; Остановить работу адаптера
        mov     AL,21h
        mov     DX,[EthBaseAddr]
        out     DX,AL
; Отобразить пакет на экране
        call    ShowPacket
; Ожидать нажатия клавиши
        MShowColorString AnyK
        call    GetChar
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
; Обработка ошибок
@@BadRegisterNumber:
        MFatalError BadRg
@@AdapterNotFound:
        MFatalError NoEth
ENDP WaitAnyPacket

;*********************************************
;* ВЫВЕСТИ НА ЭКРАН ПРИНЯТЫЙ ПАКЕТА В КОДЕ ASCII *
;*********************************************
PROC ShowPacket NEAR
        pusha
        push    ES
; Отобразить параметры принятого пакета на экране
```

```
        ; Общий размер принятого пакета в байтах
        MShowDecWord 2,21,<[word ptr DataInBuffer+2]>
        ; Адрес получателя
        MShowHexByte 3,19,<[DataInBuffer+9]>
        MShowHexByte 3,21,<[DataInBuffer+8]>
        MShowHexByte 3,23,<[DataInBuffer+7]>
        MShowHexByte 3,25,<[DataInBuffer+6]>
        MShowHexByte 3,27,<[DataInBuffer+5]>
        MShowHexByte 3,29,<[DataInBuffer+4]>
        ; Адрес отправителя
        MShowHexByte 4,19,<[DataInBuffer+15]>
        MShowHexByte 4,21,<[DataInBuffer+14]>
        MShowHexByte 4,23,<[DataInBuffer+13]>
        MShowHexByte 4,25,<[DataInBuffer+12]>
        MShowHexByte 4,27,<[DataInBuffer+11]>
        MShowHexByte 4,29,<[DataInBuffer+10]>
        ; Размер поля данных или тип пакета
        MShowHexWord 5,10,<[word ptr DataInBuffer+16]>
; Установить начало области вывода данных
        mov     AX,0B800h
        mov     ES,AX
        mov     DI,7*160
        cld
; Задать для символов светло-голубой цвет и синий фон
        mov     AH,LIGHTCYAN+BLUE*16
; Отобразить содержимое поля данных в коде ASCII
        mov     SI,offset DataInBuffer
        add     SI,18     ;адрес поля данных
        mov     CX,[RecPackSize]
        sub     CX,18     ;размер поля данных в байтах
@@OutNextChar:
        lodsb
        stosw
        loop    @@OutNextChar
        pop     ES
        popa
        ret
ENDP ShowPacket
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить процедуры перевода десятичных чисел
include "list2_05.inc"
; Подключить процедуры для обслуживания работы
; контроллера Ethernet
include "list9_01.inc"

END
```

Работа сети Ethernet начинается с установления контакта между узлами сети. Первоначально все узлы находятся в режиме прослушивания.

Процесс установления контакта между узлами сети Ethernet начинается с обмена физическими адресами. Адаптер, который первым захватил линию передачи, отправляет свой физический адрес в широковещательном режиме всем остальным узлам сегмента сети и переходит в режим приема; далее процесс повторяется, пока все адаптеры поочередно не сообщат свои физические адреса. Получив адреса всех узлов сегмента, компьютеры могут затем взаимодействовать между собой, используя эти адреса.

Листинг 9.4 содержит программу TransferTestPacket, которая производит передачу тестового пакета в широковещательном режиме. Блок данных пакета имеет размер 256 байт и содержит числа от 0 до 255. Программа использует процедуры для работы с NE2000-совместимым адаптером из листинга 9.1, а также универсальные процедуры ввода-вывода из листинга 1.2 и процедуры перевода десятичных чисел из листинга 2.5.

Программа передачи тестового пакета ориентирована на совместное использование с программой прослушивания сети из листинга 9.3. Соедините два компьютера между собой и на одном из них запустите программу прослушивания, а затем запустите программу передачи тестового пакета на другом. После передачи пакета программа прослушивания должна показать физический адрес адаптера, передавшего пакет.

ПРИМЕЧАНИЕ

В том случае, если в сеть соединено больше двух компьютеров, на время выполнения эксперимента желательно выключить питание компьютеров, не участвующих в эксперименте.

Листинг 9.4. Программа, передающая тестовый пакет данных

```
IDEAL
P386
LOCALS
MODEL MEDIUM

; Подключить файл мнемонических обозначений
; кодов управляющих клавиш и цветовых кодов
include "list1_03.inc"
; Подключить файл макросов
include "list1_04.inc"
```

```
DATASEG
; Текстовые сообщения
Txt0 DB LIGHTCYAN,0,20
     DB "ФОРМИРОВАНИЕ И ПЕРЕДАЧА ТЕСТОВОГО ПАКЕТА",0
     DB LIGHTCYAN,1,16,"ПРИ ПОМОЩИ NE2000-СОВМЕСТИМОГО "
     DB "АДАПТЕРА ETHERNET",0
     DB LIGHTGREEN,12,27,"Передача пакета завершена",0
AnyK DB YELLOW,24,29,"Нажмите любую клавишу",0
ENDS

SEGMENT sseg para stack 'STACK'
DB 400h DUP(?)
ENDS

CODESEG
;*****************************
;* Основной модуль программы *
;*****************************
PROC TransferTestPacket
        mov     AX,DGROUP
        mov     DS,AX
        mov     [CS:MainDataSeg],AX
; Установить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Скрыть курсор - убрать за нижнюю границу экрана
        mov     [ScreenString],25
        mov     [ScreenColumn],0
        call    SetCursorPosition
; Найти адаптер Ethernet
        call    SearchEthernetContr
; Проверить тип адаптера
        cmp     [EthDeviceID],8029h ;RTL8029AS
        je      @@InitAdapter
        cmp     [EthDeviceID],0926h ;VT86C926
        jne     @@AdapterNotFound
@@InitAdapter:
; Инициализировать сетевой адаптер
        call    InitializeAdapter
; Загрузить физический адрес и групповой адрес
        call    SetEthernertAddress
; Активизировать адаптер
        mov     AL,22h
        mov     DX,[EthBaseAddr]
        out     DX,AL
; Передать тестовый пакет
        call    SendTestPack
; Остановить работу адаптера
        mov     AL,21h
        mov     DX,[EthBaseAddr]
```

продолжение ⇗

Листинг 9.4 (*продолжение*)

```
        out     DX,AL
; Вывести текстовое сообщение о завершении передачи
        MShowColorText 3,Txt0
        MShowColorString AnyK
        call    GetChar
; Переустановить текстовый режим и очистить экран
        mov     AX,3
        int     10h
; Выход в DOS
        mov     AH,4Ch
        int     21h
; Обработка ошибок
@@BadRegisterNumber:
        MFatalError BadRg
@@AdapterNotFound:
        MFatalError NoEth
ENDP TransferTestPacket

;************************************************
;* СФОРМИРОВАТЬ И ПЕРЕДАТЬ ТЕСТОВЫЙ ПАКЕТ ДАННЫХ *
;* Входные параметры:                           *
;* AL - код символа-заполнителя.                *
;************************************************
PROC SendTestPack near
        pusha
; Сформировать заголовок пакета данных
        ; Адрес получателя (групповой)
        mov     [word ptr DataOutBuffer],0FFFFh
        mov     [word ptr DataOutBuffer+2],0FFFFh
        mov     [word ptr DataDutBuffer+4],0FFFFh
        ; Адрес отправителя
        mov     AL,[PhisicalAddress]
        mov     [DataOutBuffer+6],AL
        mov     AL,[PhisicalAddress+2]
        mov     [DataOutBuffer+7],AL
        mov     AL,[PhisicalAddress+4]
        mov     [DataOutBuffer+8],AL
        mov     AL,[PhisicalAddress+6]
        mov     [DataOutBuffer+9],AL
        mov     AL,[PhisicalAddress+8]
        mov     [DataOutBuffer+10],AL
        mov     AL,[PhisicalAddress+10]
        mov     [DataOutBuffer+11],AL
        ; Размер пакета в байтах
        mov     [word ptr DataOutBuffer+12],256
; Создать тестовый блок данных, заполнив его
; последовательностью чисел от 0 до 255
        mov     CX,256
        xor     AL,AL
```

```
        mov     BX,offset DataOutBuffer+14
@@NextByte:
        mov     [BX],AL
        inc     AL
        inc     BX
        loop    @@NextByte
; Загрузить пакет в память адаптера
        mov     SI,offset DataOutBuffer
        mov     CX,14+256
        call    SendPacket
; Цикл ожидания завершения передачи пакета
@@Wait: mov     DX,[EthBaseAddr]
        in      AL,DX
        cmp     AL,26h ;передатчик занят?
        je      @@Wait
        popa
        ret
ENDP SendTestPack
ENDS

; Подключить процедуры ввода данных и вывода на экран
; в текстовом режиме
include "list1_02.inc"
; Подключить процедуры для обслуживания работы
; адаптера Ethernet
include "list9_01.inc"

END
```

Соединение на основе коаксиального кабеля очень чувствительно к качеству заземления компьютеров (нарушение заземления одного системного блока может привести к повреждению всех адаптеров в сегменте сети), поэтому при проведении экспериментов с сетевыми адаптерами желательно использовать соединение на основе UTP-кабеля с витыми парами проводов.

СОВЕТ

Для соединения в сеть двух компьютеров напрямую, без использования концентратора, можно использовать так называемый перекрестный кабель (*crossover cable*) с двумя витыми парами проводов [17]. Контакты разъемов RJ-45 соединяются следующим образом: 1–3 и 2–6 (первая витая пара), 3–1 и 6–2 (вторая витая пара).

Заключение

Рекомендации по технике безопасности при проведении экспериментов на компьютере

Персональные компьютеры отличаются от других бытовых устройств широким диапазоном возможных сфер применения и высокой сложностью внутреннего устройства. Подобные свойства обеспечивают ряд преимуществ, но и создают дополнительные опасности, которые могут возникать в процессе эксплуатации и угрожать как пользователю, так и компьютеру (или подключенному к нему периферийному оборудованию). Веселые законы Мерфи [13] в реальном воплощении выглядят порой совершенно не смешно.

На основании данных, полученных из литературы, и собственного опыта я могу предложить следующие рекомендации по обеспечению безопасности оператора в процессе проведения экспериментов на компьютере.

- Необходимо соблюдать общие правила работы с электрооборудованием, то есть не лезть руками внутрь устройства, находящегося под напряжением. Прежде чем присоединять или отсоединять кабели, устанавливать в разъемы платы расширения или извлекать их, необходимо отключить компьютер и *все* присоединенное к нему оборудование от сети электропитания.
- При проектировании интерфейса пользователя нужно учитывать психологические и физиологические особенности человеческого организма. Например, для оператора крайне нежелательны яркий фон изображения, мерцание, наличие на экране большого количества движущихся или непрерывно изменяющихся объектов, нечитабельный (мелкий) шрифт и т. д. Следует также воздерживаться от злоупотребления звуковыми сигналами.
- В последнее время опять стали модными психологические опыты с мерцающими цветными изображениями и 25-м кадром. Мода, скорее всего, связана с простотой программной реализации подобных трюков, поскольку соответствующие аппаратные возможности были заложены в видеоконтроллеры персональных

компьютеров изначально, еще 20 лет назад. Однако подобные эксперименты связаны с *очень* большим риском, и проводить их допустимо только в специально оборудованных лабораториях под наблюдением врача: мерцание может вызвать эпилептический припадок, а 25-й кадр — серьезные расстройства психики. За весь период существования телевидения и компьютеров ни одна из солидных фирм так и не сумела найти 25-му кадру хоть какую-нибудь безопасную область применения.

- Любая деятельность (равно как и бездеятельность) связана с риском, а проведение эксперимента связано с повышенным риском: результат опыта нельзя точно предсказать, иначе нет необходимости в его проведении. Следовательно, экспериментатор должен обладать «правом на риск». Прежде чем проводить опыты, подумайте, обладаете ли вы подобным правом в данном месте, в данное время и по отношению к данному оборудованию. В нашей стране компьютеры являются персональными только по названию, а работают обычно в многопользовательском режиме: испортив компьютер или программное обеспечение, вы можете создать массу проблем не только себе, но и другим людям. Короче говоря, не следует ставить опыты на любимом компьютере своего шефа или на новом сервере организации, в которой вы работаете.
- Любая неисправность компьютера порождает стресс у пользователя, причем независимо от того, используется компьютер обычно для работы или развлечений. В случае полного выхода из строя любого из основных узлов стресс бывает очень сильным: ремонт стоит дорого, а вся связанная с компьютером работа откладывается до его завершения. Паника — естественная, но опасная и совершенно бесполезная форма реакции на аварийную ситуацию: кроме расходов на ремонт, приходится тратить деньги на лекарства. Когда компьютер вдруг выходит из строя, постарайтесь в первую очередь успокоиться: если вы заработаете инфаркт, ситуация лучше не станет.

Для обеспечения сохранности оборудования также можно дать определенные рекомендации.

- Перед проведением любых операций по подключению и отключению оборудования необходимо выключить компьютер, а при выполнении работы внутри корпуса — вообще отключить компьютер от электросети. Горячую замену при включенном напряжении питания допускают только два устройства — клавиатура,

подключаемая к стандартному разъему, и мышь, подключаемая к последовательному порту. Все остальные устройства, в том числе мышь типа PS/2, могут получить повреждения при присоединении их к включенному компьютеру или повредить системную плату.

- Параллельные и последовательные порты компьютера не имеют электрической развязки, поэтому крайне нежелательно подключать к ним какие-либо самодельные устройства, особенно имеющие собственные источники электропитания. Ошибка при подключении обычно выводит из строя блок питания компьютера, а иногда и системную плату.
- Эксперименты по подключению самодельного оборудования следует по возможности проводить на специальном стенде, собранном из старых, но исправных комплектующих, оставшихся после модернизации. Количество разнообразных ошибок, возникающих на этапе проектирования (из-за неточностей в документации) и монтажа (из-за невнимательности), обычно довольно велико, а покупать каждый раз новую системную плату — весьма накладно.
- Соединение нескольких компьютеров в сеть при помощи коаксиального кабеля требует наличия в здании заземления и строгого соблюдения правил его выполнения. Помните, что на корпусе незаземленного компьютера присутствует переменное напряжение 110 В (конденсаторы фильтра высокочастотных помех блока питания в этом случае выступают в качестве делителя сетевого напряжения).
- Эксперименты с устройствами для хранения информации, то есть с дисководами, следует начинать с выполнения операций считывания данных. Переходить к записи данных можно только после освоения правил работы с дисководом — любая ошибка при выполнении записи может привести к потере информации на диске, причем очень часто невозможно определить, какая конкретно информация была повреждена. Эксперименты по форматированию жестких дисков проводить вообще не желательно — команды форматирования недостаточно стандартизированы. Если есть возможность, на начальном этапе проведения опытов лучше использовать какой-нибудь старый дисковод, с которым в случае чего будет не жалко расстаться.
- Нежелательно проводить эксперименты с параметрами видеорежимов на уровне регистров видеоконтроллера — пользы от та-

ких опытов обычно никакой, а вероятность сжечь монитор достаточно велика (кстати, из всех компонентов персонального компьютера монитор является сейчас самым дорогостоящим).

- В процессе экспериментов с сетевыми адаптерами желательно использовать для соединения адаптеров кабель UTP, так как в этом случае значительно меньше риск повреждения оборудования при нарушении заземления системных блоков.
- Использование флэш-памяти в качестве ПЗУ — интересный, но крайне рискованный прием проектировщиков. Попытка самостоятельной перезаписи флэш-памяти (даже с помощью специальных утилит, поставляемых изготовителем) — самый простой и надежный способ вывести из строя многие типы устройств. В настоящее время перезаписываемой памятью снабжены процессоры, материнские платы, видеоконтроллеры, модемы, дисководы. Соответственно, многие важные компоненты компьютера теперь уязвимы как для грубых вирусов, так и для некорректных действий пользователя.
- Желательно, чтобы компьютер был защищен от неисправностей в сети электропитания. Простейшей защитой от помех и высоковольтных импульсов являются сетевые фильтры, а для защиты от бросков напряжения и внезапного отключения питания служат источники бесперебойного питания (UPS); лучше всего, если защитное устройство является комбинированным, то есть выполняет функции и фильтра, и UPS. Наличие защиты особенно важно при экспериментах с запоминающими устройствами — отключение питания в момент перезаписи Flash-BIOS некоторого устройства (например, системной платы) делает устройство полностью неработоспособным, причем устранить это повреждение обычно можно только в центре технического обслуживания. При записи информации на гибкие диски и компакт-диски сбои питания часто приводят к повреждению носителей информации (дисков). Провал или бросок напряжения при перезаписи начального сектора жесткого диска может сделать его непригодным для дальнейшего использования (в результате недоступности начального сектора для считывания и записи), а вся имеющаяся на диске информация будет потеряна.
- Бытовые персональные компьютеры стоят не слишком дорого, поэтому их часто используют при создании *макетов* измерительных и управляющих устройств. На таком макете можно отрабатывать программное обеспечение и обучать персонал (опера-

торов), однако применять бытовой ПК для управления реальными объектами *не рекомендуется* — он не обладает достаточной надежностью и обычно не соответствует требованиям окружающей среды (промышленные управляющие устройства должны сохранять работоспособность в широком диапазоне температур при наличии вибрации, пыли и т. д.). Сбой в процессе управления никак не сказывается на самом компьютере, но может повредить управляемый объект, стоимость которого обычно на несколько порядков превышает стоимость ПК.

Желаю удачи!

Кулаков Владимир

Адрес электронной почты: uiits@miem.edu.ru.

Литература

1. *Абраш М.* Таинства программирования графики. — К.: ЕвроСИБ, 1996.
2. *Богумирский Б. С.* Руководство пользователя ПЭВМ. — СПб.: Ассоциация OILCO, 1992.
3. *Браун Р., Кайл Дж.* Справочник по прерываниям для IBM PC. — М.: Мир, 1994.
4. *Григорьев В. Л.* Архитектура и программирование арифметического сопроцессора. — М.: Энергоатомиздат, 1991.
5. *Григорьев В. Л.* Микропроцессор i486. Архитектура и программирование. — М.: ГРАНАЛ, 1993.
6. *Гук М.* Аппаратные средства IBM PC: Энциклопедия. — СПб.: Питер Ком, 1999.
7. *Гук М.* Интерфейсы ПК: справочник. — СПб.: Питер, 1999.
8. *Гуртовцев А. Л., Гудыменко С. В.* Программы для микропроцессоров. — Минск.: Высш. шк., 1989.
9. *Гюнтер Борн.* Форматы данных. — СПб.: BHV, 1995.
10. *Данкан Р.* Профессиональная работа в MS DOS. — М.: Мир, 1993.
11. *Джордейн Р.* Справочник программиста персональных компьютеров типа IBM PC, XT и AT. — М.: Финансы и статистика, 1992.
12. Закон Мерфи. — Минск: ООО «Попурри», 1997.
13. Использование Turbo Assembler при разработке программ. — Киев: Диалектика, 1994.
14. *Лукач Ю. С., Сибиряков А. Е.* Архитектура ввода-вывода персональных ЭВМ IBM PC. — Свердловск: Инженерно-техническое бюро, 1990.
15. *Малиновский Б. Н.* История вычислительной техники в лицах. — Киев: Фирма «КИТ», 1995.
16. Микропроцессорный комплект К1810. — М.: Высшая школа, 1990.
17. *Новиков Ю. В., Карпенко Д. Г.* Аппаратура локальных сетей: функции, выбор, разработка. — М.: ЭКОМ, 1998.
18. Однокристальные микроЭВМ. — М.: МИКАП, 1994.
19. *Олифер В. Г., Олифер Н. А.* Компьютерные сети. Принципы, технологии, протоколы. — СПб.: Питер, 2001.
20. *Орлов А.* Анимация в компьютерных играх.: Мир ПК 1993, №1.
21. *Паппас К., Марри У.* Микропроцессор 80386: Справочник. — М.: Радио и связь 1993.
22. Персональные ЭВМ на основе архитектуры Intel 80386. — Обнинск: ИНВЕС КО, 1993.
23. Печатающие устройства для персональных ЭВМ: Справочник. — М.: Радио и связь, 1992.
24. Разработка боевых программ в НИИ-5. // PC WEEK/RE 1999, № 40.

25. *Романов В. Ю.* Популярные форматы файлов для хранения графических изображений на IBM PC. – М.: Унитех, 1992.
26. *Синев А.* Как создать оконный интерфейс. // КомпьютерПресс 1991, № 1,2.
27. Техника программирования на Turbo C. – М.: «И.В.К. – СОФТ», 1991.
28. *Том Сван.* Форматы файлов Windows. – М.: БИНОМ, 1994.
29. *Уилтон Р.* Видеосистемы персональных компьютеров IBM PC и PS/2. Руководство по программированию. – М.: Радио и связь, 1994.
30. *Фролов А. В., Фролов Г. В.* Аппаратное обеспечение IBM PC. – М.: ДИАЛОГ-МИФИ, 1992.
31. *Фролов А. В., Фролов Г. В.* Защищенный режим процессоров Intel 80286, 80386, 80486: Практическое руководство по использованию защищенного режима. – М.: ДИАЛОГ-МИФИ, 1992.
32. *Фролов А. В., Фролов Г. В.* Операционная система MS DOS. – М.: ДИАЛОГ-МИФИ, 1991.
33. *Фролов А. В., Фролов Г. В.* Программирование видеоадаптеров CGA, EGA и VGA. – М.: ДИАЛОГ-МИФИ, 1992.
34. *Юров В. И.* Assembler. – СПб.: Питер, 2000.
35. AN519. Implementing a Simple Serial Mouse Controller. – Microchip Technology Inc., 1997.
36. AT/LANTIC Software Developer's Guide. – National Semiconductor Corp., 1993.
37. ATA Host Adapter Standards. Revision 0b. T13/1510D. – ANSI, 2001.
38. Avenger Super High Performance Graphics Engine for 3D Game Acceleration. – 3Dfx Interactive Inc., 1999.
39. Craig Peacock. USB in a Nutshell. Making Sense of the USB Standard. – Beyondlogic, 2002.
40. Dave Williams. Programmer's Technical Reference for MSDOS and the IBM PC. – 1989.
41. DP8390D/NS32490D NIC - Network Interface Controller. – National Semiconductor Corp., 1995.
42. DP83901A SNIC Serial Network Interface Controller. – National Semiconductor Corp., 1995.
43. DP83905 AT/LANTIC Local Area Network Twisted-Pair Interface Controller. – National Semiconductor Corp., 1995.
44. Enhanced Parallel Port BIOS Specification. – FarPoint Communications, 1993.
45. EPSON ESC/P Reference Manual. – Seiko Epson Corp., Nagano, Japan, 1997.
46. EPSON FX series printer. User's Manual. – Seiko Epson Corp., Nagano, Japan, 1988.
47. EPSON LQ 850/950/1050. User's Manual. – Seiko Epson Corp., Nagano, Japan, 1988.
48. EPSON Programming Guide For 4 Color EPSON Stylus COLOR Ink Jet Printers: Stylus COLOR 440, Stylus COLOR 640, Stylus COLOR 740, Stylus COLOR 900. – EPSON Imaging Technology Center, 1999.
49. EPSON Programming Guide For 4 Color EPSON Stylus Ink Jet Printer: EPSON Stylus C20SX, EPSON Stylus C20UX, EPSON Stylus C40SX, EPSON Stylus C40UX. – EPSON Imaging Technology Center, 2001.

50. EPSON Programming Guide For 4 Color EPSON Stylus Ink Jet Printer Stylus C60. — EPSON Imaging Technology Center, 2001.
51. Extended Capabilities Port: Specification. — Microsoft Corp., 1993.
52. FAT: General Overview of On-Disk Format. — Microsoft Corporation, 1999.
53. FDC37C665GT, FDC37C666GT High-Performance Multi-Mode Parallel Port Super I/O Floppy Disk Controllers. — Standard Microsystems Corp., 1994.
54. HP DeskJet 600/800 Series Printers Software Developer's PCL Guide. — Hewlett Packard, 1997.
55. HT6513B M+ Plug & Play Mouse Controller. — Holtek Semiconductor Inc., 2000.
56. HT6523 PS/2 Mouse Controller. — Holtek Semiconductor Inc., 1996.
57. HT82K28A Win98 Keyboard Encoder. — Holtek Semiconductor Inc., 1999.
58. HT82M33A 3D Mouse Controller. — Holtek Semiconductor Inc., 1999.
59. HT82M398A WIN2000 3D PS/2 Mouse Controller. — Holtek Semiconductor Inc., 2000.
60. HT82M39A 3D PS/2 Mouse Controller. — Holtek Semiconductor Inc., 1999.
61. IEEE Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers. — IEEE, 2000.
62. Information Technology — AT Attachment-3 Interface (ATA-3), Revision 7b, X3T10/2008D. — ANSI, 1997.
63. Information Technology — AT Attachment with Packet Interface-6 (ATA/ATAPI-6), Revision 3b, T13/1410D. — ANSI, 2002.
64. Information Technology — BIOS Enhanced Disk Drive Service (EDD), Revision 5, T13/1386D. — ANSI, 2000.
65. Intel 810 Chipset Family Programmer's Reference Manual. — Intel Corp., 1999.
66. Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture. — Intel Corp., 1999.
67. Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference. — Intel Corp., 1999.
68. Intel Architecture Software Developer's Manual, Volume 3: System Programming. — Intel Corp., 1999.
69. KBD42W11 Keyboard Controller. — Standard Microsystems Corp.
70. KBD43W13 Keyboard and PS/2 Mouse Controller. — Standard Microsystems Corp.
71. Keyboard Scan Code Specification. — Microsoft Corp., 2000.
72. Larry Smith. Accessing high memory. — Programmer's Journal, May/June 1990.
73. Lexmark Technical Reference. — Lexmark International, Inc., 1999.
74. Long Filename Specification. — Microsoft Corporation, 1992.
75. Matrox MGA-G200 Specification. — Matrox Graphics Inc., 1998.
76. Matrox MGA-G400 Specification. — Matrox Graphics Inc., 1999.
77. Michael Abrash. Measuring Performance. — Programmer's Journal 7.4, 1989.
78. Michael Abrash. VGA Text Paging. — Programmer's Journal 6.5, 1988.
79. MultiKey/42i Developer's Technical Manual. — Phoenix Technologies Ltd., 1996.
80. PC 2001 System Design Guide. — Intel Corporation and Microsoft Corporation, 2000.

81. PCI BIOS Specification. Revision 2.1. – PCI Special Interest Group, 1994.
82. PCI Local Bus Specification. Revision 2.2. – PCI Special Interest Group, 1998.
83. PCL 5 Color Technical Reference Manual. – Hewlett Packard, 1994.
84. PCL 5 Printer Language Technical Reference Manual. – Hewlett Packard, 1992.
85. PhoenixBIOS 4.0 User's Manual. – Phoenix Technologies Ltd., 2000.
86. Plug and Play Parallel Port Devices. – Microsoft Corp., 1993.
87. Printer Job Language Technical Reference Manual. – Hewlett Packard, 1997.
88. RTL8029AS Realtek PCI Full-Duplex Ethernet Controller with built-in SRAM. – Realtek Semiconductor Corp., 1997.
89. The PCL Implementor's Guide. Version 6.0. – Hewlett Packard, 1995.
90. Thomas Roden. Four Gigabytes in Real Mode. – Programmer's Journal 7.6, 1989.
91. Universal Host Controller Interface (UCHI) Design Guide. Revision 1.1. – Intel Corp., 1996.
92. Universal Serial Bus Device Class Definition for Human Interface Devices (HID). – USB Implementers Forum, 1999.
93. Universal Serial Bus Device Class Definition for Printing Devices. – USB Implementers Forum, 1997.
94. Universal Serial Bus Language Identifiers. – USB Implementers Forum, 2000.
95. Universal Serial Bus Specification. Revision 1.1. – Compaq Computer Corp., Intel Corp., Microsoft Corp., NEC Corp., 1998.
96. VESA BIOS EXTENSION Core Function Standard, Version 3.0. – Video Electronics Standards Association, 1998.
97. VIA VT86C926 Amazon PCI Ethernet Controller Data Sheet. – VIA Technologies Inc., 1995.
98. VT82C42 Keyboard Controller. – VIA Technologies, Inc., 1995.
99. Writing Drivers for the DP8390 NIC Family of Ethernet Controllers. – National Semiconductor Corp., 1993.

Англоязычная документация получена в основном из Интернета с серверов фирм-изготовителей оборудования:

Hewlett-Packard. – www.hp.com

Holtek Semiconductor Inc. – www.holtek.com.tw

Intel Corp. – developer.intel.com

Matrox Graphics Inc. – www.matrox.com

Microchip Technology Inc. – www.micrichip.com

Microsoft Corp. – www.microsoft.com

National Semiconductor Corp. – www.national.com

PCI Special Interest Grope – www.pcisig.com

Phoenix Technologies Ltd. – www.phoenix.com, www.ptltd.com

Realtek Semiconductor Corp. – www.realtek.com.tw

Seiko Epson Corp. – www.epson.com, www.epsondevelopers.com

Standard Microsystems Corp. – www.smsc.com

VIA Technologies, Inc. – www.via.com.tw, www.viatech.com

Алфавитный указатель

Символы

A

B

C

D

E

F

H

I

L

M

Л

М

Н

Х